Un modèle d'interaction en entrée pour des systèmes interactifs multi-dispositifs hautement configurables

Pierre Dragicevic

UNIVERSITÉ DE NANTES

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES
DE L'INFORMATION ET DES MATÉRIAUX

Année : 2004 N^0 B.U. :

Thèse de Doctorat de l'Université de Nantes

Spécialité : INFORMATIQUE

Présentée et soutenue publiquement par

Pierre DRAGICEVIC

le 9 mars 2004

à l'École Nationale Supérieure
des Techniques Industrielles et des Mines de Nantes

Un modèle d'interaction en entrée pour des systèmes interactifs multi-dispositifs hautement configurables

Jury :

Président	:	**Henri BRIAND**	Professeur, Université de Nantes
Rapporteurs	:	**Michel BEAUDOUIN-LAFON**	Professeur, Université Paris-Sud
	:	**Philippe PALANQUE**	Professeur, Université Toulouse 3
Examinateur	:	**Stéphane CHATTY**	Intuilab, Toulouse

Directeur de thèse	:	**Gérard HÉGRON**
Laboratoire	:	CERMA, CNRS UMR 1563, Université de Nantes
Co-encadrant	:	**Jean-Daniel FEKETE**
Laboratoire	:	INRIA Unité de Recherche Futurs

N^o ED 0366-

Table des matières

Introduction

D’autres manières de contrôler nos ordinateurs

L’informatique n’a cessé de progresser depuis les premiers calculateurs jusqu’aux micro-ordinateurs modernes. Aujourd’hui, si les machines de nos foyers continuent à voir leur puissance de calcul et leurs capacités graphiques croître d’année en année, leur évolution *paraît* avoir atteint un point satisfaisant sous bien d’autres aspects. En particulier, nous sommes tous habitués à taper sur un clavier et à cliquer sur des boutons, ouvrir des menus ou déplacer des icônes avec une souris : notre *façon de contrôler* les ordinateurs s’est standardisée, et rien ne semble apparemment en voie de compromettre ces standards sur lesquels s’accordent à la fois les fabricants de matériel, les éditeurs de logiciels et la plupart des utilisateurs.

Il existe néanmoins bien d’autres manières de contrôler des applications interactives. Les manettes de jeu sont d’usage courant, et les ordinateurs de poche modernes sont en train de démocratiser l’emploi du stylet. Des dispositifs d’entrée moins connus tels que les tablettes graphiques, les instruments de musique électroniques ou les contrôleurs 3D sont depuis longtemps exploités dans le monde professionnel ou semi-professionnel. Enfin, les utilisateurs pour lesquels l’usage d’un clavier ou d’une souris est difficile voire impossible emploient également des dispositifs d’entrée spécialisés.

Au-delà de leur grande variété, les dispositifs d’entrée alternatifs possèdent tous un point commun : ils sont particulièrement adaptés à un contexte donné, c’est-à-dire à certains types d’utilisateurs (handicapés moteurs, par exemple), de tâches (jeux vidéo ou conception graphique) ou d’environnements de travail (environnement mobile). Ces dispositifs non conventionnels sont indispensables dans la mesure où les dispositifs standard ne sont pas toujours pertinents. De nombreux travaux de recherche ont de plus montré qu’il était *toujours* possible de rendre l’interaction plus naturelle et plus productive par l’usage de dispositifs d’entrée dédiés au contexte.

Parmi les nombreux dispositifs d'entrée existants, le clavier et la souris bénéficient d'un statut particulier en tant que dispositifs *génériques*. Ils ont été conçus pour être efficaces dans la plupart des situations courantes : un environnement de travail de type bureau, un utilisateur « moyen », et des tâches courantes comme la saisie d'un courrier ou la navigation sur Internet. Or ces dispositifs sont en réalité peu efficaces, quelle que soit la tâche, car ils n'exploitent que partiellement nos capacités motrices et nous obligent à des manipulations séquentielles plutôt que parallèles.

Nous utilisons nos deux mains dans presque toutes nos tâches quotidiennes et sommes par ailleurs capables d'apprendre à coordonner efficacement un grand nombre de groupes musculaires (pour jouer d'un instrument ou conduire une voiture, par exemple). Nos doigts possèdent également des capacités motrices exceptionnelles. Sur un ordinateur, ces capacités sont assez bien exploitées pour la saisie textuelle mais très peu pour la manipulation d'objets graphiques. La parole constituerait également un complément efficace au contrôle moteur. Ces idées ont été depuis longtemps confirmées par des travaux de recherche, qui montrent qu'une interaction réellement efficace passe par la *multiplication* et la *diversification* des dispositifs d'entrée.

Si la qualité de l'interaction repose pour beaucoup sur la partie « physique » de l'interface, elle dépend également de la façon dont les dispositifs physiques sont exploités au niveau logiciel. Actuellement, utiliser une souris est synonyme de cliquer, double-cliquer ou effectuer des cliquer-glisser sur des boutons, barres de défilement et autres menus. Le vocabulaire est limité, et les éléments graphiques standardisés ont montré leurs faiblesses : ils entravent la tâche réelle (rédaction d'un document, par exemple) en occupant de la place et en imposant des déplacements répétés de l'attention visuelle. Là encore, de nouvelles approches, visant à rendre l'interaction plus *concise* et plus *directe*, ont été proposées dans bon nombre de travaux de recherche, et prouvent que même l'interaction à la souris est susceptible d'être considérablement améliorée.

Des systèmes interactifs encore figés

Les standards en Interaction Homme-Machine sont souhaitables car il en découle une réutilisabilité qui profite aux programmeurs d'applications, et une cohérence qui profite, dans une certaine mesure, aux utilisateurs. Nos interfaces actuelles, avec leurs souris, icônes et fenêtres s'inspirent principalement du système *Xerox Star*, initialement issu de la recherche. Cependant, ce système est maintenant vieux de plus de vingt ans et la recherche en IHM a eu le temps d'explorer, mettre en œuvre et tester des méthodes d'interaction alternatives adaptées aussi bien aux situations les plus courantes qu'à des contextes très spécifiques.

Ces recherches montrent à quel point les interfaces dites modernes sont, du point de vue de l'utilisabilité, au mieux inefficaces, au pire totalement inadaptées. Elles prouvent également que pour une interaction plus naturelle, productive, personnalisable et accessible aux utilisateurs handicapés, il est essentiel que nos systèmes informatiques puissent être librement contrôlés avec des dispositifs d'entrée multiples et variés, d'une manière qui prenne en compte les spécificités des dispositifs et les besoins de l'utilisateur.

Malheureusement, les standards ont si profondément structuré les systèmes interactifs qu'il est maintenant difficile de s'en défaire. Alors que certains dispositifs d'entrée non conventionnels deviennent de plus en plus abordables (tablettes graphiques, webcams, microphones) et de plus en plus faciles à installer, leur prise en charge effective par les systèmes interactifs reste quasi-inexistante : la plupart des dispositifs sont ignorés par la majorité des applications (les manettes ne servent que dans

les jeux), et les autres sont sous-exploités (la tablette graphique est vue comme une souris, sa haute précision et sa sensibilité à la pression sont occultées). Réciproquement, une application est très difficile, voire impossible à utiliser lorsqu'un dispositif standard manque, ce qui constitue un problème majeur pour l'accessibilité aux handicapés et la portabilité des applications entre les plate-formes de différente nature.

Les applications interactives sont excessivement figées du point de vue de l'interaction parce qu'elles sont construites avec des outils qui sont tous câblés pour une utilisation exclusive et stéréotypée d'un clavier et d'une souris. La prise en charge de dispositifs d'entrée non conventionnels est en conséquence extrêmement difficile à mettre en œuvre par les programmeurs. Les innovations proposées par la recherche, comme l'interaction gestuelle ou vocale, sont pour les mêmes raisons très rarement employées dans les applications commerciales.

Une problématique de recherche réelle

Un des objectifs de la recherche en IHM consiste à concevoir, prototyper et déterminer l'utilisabilité d'interfaces novatrices, approche qui a donné de nombreux résultats, dont venons de tirer des conclusions. Un autre objectif, essentiel et complémentaire, vise à produire des modèles et des outils qui facilitent la construction de ce type d'interfaces. La tâche est loin d'être triviale et malgré des avancées importantes, les résultats obtenus ne sont pas à la hauteur des espérances.

Il a fallu déjà consacrer une bonne partie des efforts à la compréhension et à la modélisation des interfaces conventionnelles, pour lesquelles nous ne disposons toujours pas de modèle et d'outil de construction idéal. Les recherches se donnant pour but de faciliter la construction d'interfaces non conventionnelles constituent quant à elles un domaine encore très jeune. En outre, les aspects « sorties » de l'interaction (qui concernent principalement la représentation d'objets abstraits à destination de l'utilisateur et la prise en charge d'effets graphiques sophistiqués) ont assez curieusement suscité bien plus de travaux que les aspects « entrées » (qui décrivent la manipulation de ces objets par l'utilisateur), pourtant essentiels.

Dans les travaux portant sur les entrées non conventionnelles, l'intérêt s'est principalement focalisé sur la mise en œuvre d'une communication langagière avec des interfaces « intelligentes », en abordant des problèmes de haut niveau tels que l'interprétation des intentions de l'utilisateur à partir de sources d'information hétérogènes ou la prise en charge de l'ambiguïté. Malgré la complexité de ces problèmes et l'apport indéniable des contributions existantes, trop d'approches continuent de s'appuyer sur des modèles d'entrée vieux de plusieurs décennies ou à en reproduire les mêmes hypothèses simplificatrices, et mènent invariablement à des systèmes plus évolués mais tout aussi figés.

Actuellement, aucun modèle et aucun outil ne permet d'exploiter toute la richesse offerte par les dispositifs d'entrée et n'offre assez de flexibilité pour pouvoir prendre en compte des contextes d'utilisation très divers.

Les objectifs de cette thèse

Notre objectif dans cette thèse est de proposer des modèles et des outils permettant de construire des systèmes interactifs entièrement « ouverts » en entrée, c'est-à-dire des systèmes qui puissent être contrôlés avec une grande variété de dispositifs d'entrée, d'une manière qui permette d'exploiter au mieux leurs capacités.

Nous pensons qu'une approche moderne se doit de mettre l'accent sur une interaction à la fois riche et accessible, en encourageant l'usage simultané de dispositifs physiques multiples tout en offrant une prise en charge adaptée pour les entrées appauvries, c'est-à-dire plus limitées que les dispositifs standard. Nous prônons également des systèmes évolutifs et configurables, c'est-à-dire non limités à des classes figées de dispositifs et de modèles d'interaction connus, et qu'il est possible d'adapter finement aux particularités de l'utilisateur, de la tâche et de l'environnement.

Nos contributions

Dans notre thèse, nous introduisons et motivons un modèle original pour la description et la gestion de l'interaction en entrée, qui s'appuie sur la notion de *configurations d'entrée*. Une configuration d'entrée décrit la façon dont des dispositifs physiques sont connectés à une application à travers des adaptateurs. En remplaçant l'approche événementielle traditionnelle par un paradigme à flot de données réactif, ce modèle rend la gestion des entrées entièrement explicite, du dispositif physique jusqu'à l'application. Il encourage en outre la description d'interactions novatrices et hautement concurrentes.

Le modèle des configurations d'entrée a inspiré le développement du système ICON (Input Configurator), qui permet de construire des applications interactives entièrement configurables en entrée et capables d'exploiter des dispositifs et des techniques d'interaction conventionnels ou non. L'éditeur interactif d'ICON donne la possibilité aux développeurs de prototyper et de tester un grand nombre de configurations d'entrée potentielles afin de rendre leurs applications adaptées à des situations spécifiques d'entrées appauvries ou enrichies. L'utilisateur final sélectionne la configuration d'entrée qui correspond à son matériel et à ses préférences personnelles, et peut également, selon son degré d'expertise, personnaliser cette configuration pour l'adapter à ses besoins.

Organisation de ce mémoire

Dans le **chapitre 1 - L'interaction en entrée non standard**, nous donnons un large aperçu des dispositifs d'entrée non conventionnels existants et des contextes dans lesquels ils sont employés, et présentons les principales techniques connues permettant d'interagir avec les applications de façon plus efficace et plus naturelle. Nous montrons ensuite en quoi il est essentiel que nos applications puissent être contrôlées de diverses manières, pour une interaction de meilleure qualité et mieux adaptée à des utilisateurs, des environnements et des tâches très variés.

Dans le **chapitre 2 - Modèles et outils pour l'interaction en entrée**, nous détaillons l'état actuel de la recherche du point de vue de l'interaction en entrée. Nous présentons les principaux modèles d'interface et les outils de développement expérimentaux, et déterminons en quoi ils constituent ou non une avancée par rapport aux modèles et aux outils conventionnels. Nous concluons par les apports respectifs de chacune de ces contributions.

Dans le **chapitre 3 - Introduction au modèle des configurations d'entrée**, nous présentons notre modèle pour la description de l'interaction en entrée, après avoir développé et motivé du point de vue conceptuel les principales notions sur lesquelles il repose.

Dans le **chapitre 4 - La boîte à outils ICON**, nous décrivons le système ICON, développé dans le but de valider et d'affiner notre modèle dans un contexte réel de développement. Après avoir présenté cet outil et ses principales fonctionnalités, nous décrivons et commentons un certain nombre

d'exemples d'interactions non conventionnelles qu'il a permis de décrire. Nous concluons par les applications actuelles d'ICON.

Dans le **chapitre 5 - Discussion**, nous tentons de répondre aux nombreuses questions posées par notre approche, en identifiant tout d'abord les apports essentiels d'ICON et de son modèle sous-jacent, ainsi que leurs limites. Nous affinons et généralisons par ailleurs quelques aspects de notre approche, sur la base des enseignements que nous avons pu tirer de notre utilisation d'ICON. Pour finir, nous positionnons nos travaux par rapport aux contributions existantes et développons quelques perspectives.

Dans les **annexes A, B et C**, nous détaillons les mécanismes à la base du modèle des configurations d'entrée.

La lecture de ce mémoire de thèse ne devrait pas nécessiter de connaissances théoriques particulières en IHM. Une certaine pratique dans le développement d'interfaces est cependant requise pour aborder la plupart des discussions, en particulier dans le chapitre 2 et le chapitre 5. Les annexes réclament des connaissances mathématiques et algorithmiques très minimales. Le chapitre 1, et dans une moindre mesure le chapitre 3, n'appellent aucun pré-requis et suffisent à se faire une idée très générale du contexte et de notre approche.

Chapitre 1

L'interaction en entrée non standard

Sommaire

1.1 Introduction

L'objectif de ce chapitre est de donner au lecteur un aperçu de la grande variété des dispositifs d'entrée, produits commerciaux ou prototypes de recherche, et des contextes dans lesquels leur emploi est pertinent. Nous décrivons ensuite, de façon plus générale, de nouveaux moyens d'interagir avec les applications, de manière plus naturelle et efficace. Puis, nous montrons l'intérêt d'exploiter ces diverses techniques dans nos applications conventionnelles, et plus généralement, pourquoi il est essentiel qu'un système interactif puisse s'adapter à n'importe quel type d'entrée. Nous finissons par montrer en quoi nos ordinateurs actuels ne répondent pas à cette attente.

Avant de commencer, nous présentons rapidement notre domaine d'étude en introduisant les principaux termes qui nous seront utiles par la suite.

1.1.1 L'interaction en entrée et son vocabulaire

Certains termes et concepts que nous allons introduire dans cette section sont dans la littérature scientifique plutôt équivoques. Notre objectif n'est pas de répertorier et de confronter les définitions existantes ni de prétendre à l'exactitude, mais bien de proposer un vocabulaire minimal cohérent qui nous sera utile pour structurer la suite de notre discours.

L'interaction en entrée

Il est courant de distinguer dans l'interaction homme-machine les *entrées* (flux d'informations allant de l'homme vers la machine) des *sorties* (flux d'informations allant de la machine vers l'homme).

Cette distinction est parfois critiquée, à raison, sur la base d'un lien étroit entre les entrées et les sorties [HINCKLEY *et al.* 03] : en premier lieu, tout contrôle nécessite un *feedback* ou *retour utilisateur*[1] (une souris sans pointeur n'est pas utilisable). Ensuite, la façon dont sont représentés les objets de l'application joue considérablement sur la façon dont ils peuvent être contrôlés (une représentation graphique, en particulier, encouragera les techniques de pointage).

Malgré tout, il apparaît clairement dans la littérature scientifique que si certains domaines s'attachent principalement à la *représentation* de données, d'autres problématiques et d'autres aspects de l'interaction sont concernés par leur *manipulation* [BUXTON 86B]. Ce sont ces derniers aspects que recouvre le terme d'*interaction en entrée*, et que nous décrivons par la suite.

Composantes de l'interaction en entrée

Dispositif d'entrée. Un dispositif d'entrée est un appareil physique contrôlé par l'utilisateur et qui émet des informations vers l'ordinateur. Ceci inclut les périphériques comme le clavier et la souris, et exclut les périphériques de stockage[2]. Pour éviter certaines lourdeurs, nous emploierons parfois le terme simplifié de *dispositif*. (Synonymes : *périphérique d'entrée*, *périphérique de saisie*.)

[1] *Rétroaction* est peut-être la traduction la plus exacte de *feedback*, mais elle est rarement employée en IHM. Dans ce domaine, *feedback* signifie en général *user feedback*, qui se traduit assez bien par *retour utilisateur*. Cependant, nous préférerons le raccourci de *feedback* au simple mot *retour*.

[2] En anglais, le terme *human input device (HID)* est parfois préféré à *input device* lorsqu'une confusion est possible.

Technique d'interaction. Une technique d'interaction décrit *la manière de se servir d'un dispositif d'entrée* pour accomplir une tâche sur l'ordinateur [FOLEY *et al.* 90]. Par exemple, un fichier peut être supprimé à la souris, soit à l'aide d'un menu contextuel, soit par glisser-déposer dans la corbeille. En pratique, une technique d'interaction *transforme, interprète* des données et produit le *feedback* nécessaire à l'utilisateur. Les techniques d'interaction sont le plus souvent caractérisées de façon *abstraite et générique* : par exemple, le menu contextuel et le glisser-déposer ne sont pas particulièrement liés à un dispositif spécifique[3], ni à la tâche spécifique de suppression de fichiers. (Synonymes : *méthode d'interaction, technique d'entrée*).

Paradigme d'interaction. Un paradigme d'interaction est un ensemble cohérent de techniques d'interaction qui coopèrent de façon étroite, ou qui reposent sur les mêmes principes techniques ou conceptuels. À titre d'exemple, le paradigme d'interaction WIMP[4] regroupe des techniques comme le pointage et les menus. Quand au paradigme de l'interaction bimanuelle, il désigne des techniques d'interaction exploitant l'usage simultané des deux mains. (Synonyme : *style d'interaction.*)

L'interaction standard

L'**interaction standard** décrit l'ensemble des paradigmes d'interaction actuellement employés sur les micro-ordinateurs grand public pour permettre un contrôle *générique* des applications par l'usage des **dispositifs d'entrée standard** que sont le clavier et la souris.

La liste des paradigmes d'interaction standard qui suit est inspirée du traditionnel historique des styles d'interaction [PREECE *et al.* 94], dans lequel nous avons remplacé la manipulation directe par le paradigme d'interaction WIMP :

- **Les langages de commande :** Interfaces textuelles du type MS-DOS ou terminal Unix où des commandes sont saisies au clavier. Les différentes commandes possibles ne sont pas explicites et nécessitent une mémorisation.
- **Les systèmes de menus :** Un menu est la présentation d'une liste de choix dans laquelle l'utilisateur opère une sélection au clavier ou par pointage. Une *boîte de dialogue* du type "Êtes-vous sûr ? oui/non" est assimilable à un menu. Un *assistant* est une succession de menus où la navigation est contrôlée par le système.
- **Les formulaires :** Les formulaires à compléter étendent les systèmes de menus en ajoutant des *champs* de texte qu'il est possible de renseigner. Les *tableurs* en sont une version fonctionnellement plus sophistiquée.
- **L'interaction WIMP** : L'interaction WIMP est une version standardisée et appauvrie du paradigme de la *manipulation directe*, qui décrit des interfaces donnant l'illusion de travailler directement sur les objets de la tâche plutôt que de transmettre des commandes à une machine [SHNEIDERMAN 83, HUTCHINS *et al.* 86] (ce modèle d'interaction sera décrit dans la section 2.5.1 page 56). Le paradigme de la manipulation directe ne constitue pas un standard mais décrit les caractéristiques idéales de l'interaction graphique. Nous désignons l'interaction WIMP comme étant une combinaison des trois paradigmes d'interaction suivants :
  - **Le glisser-déposer**[5] **:** La métaphore du bureau, popularisée par les systèmes Macintosh, est

[3] D'autres dispositifs que la souris permettent d'effectuer des tâches similaires. Nous en évoquerons quelques-uns plus loin.

[4] WIMP : Windows, Icons, Mouse, Pull-down menus (fenêtres, icônes, souris, menus déroulants). Terme employé pour désigner les interfaces d'aujourd'hui, parfois dans un sens péjoratif.

[5] En anglais, *drag-and-drop*. Le cliquer-glisser désigne le geste lui-même.

souvent considérée comme l'exemple-type de la manipulation directe : les fichiers sont des objets graphiques nommés *icônes* et les commandes (déplacer, effacer) s'effectuent principalement par des *cliquer-glisser* d'une icône vers un objet sensible (autre icône ou conteneur). Cette technique est également employée comme mécanisme de communication entre les fenêtres.

– **Le fenêtrage :** Les *fenêtres* sont des zones rectangulaires de l'écran qui sont agençables et superposables. L'utilisateur choisit la fenêtre *active* qui sera l'objet de l'interaction. Les *documents* dans les applications multi-documents reposent sur le même principe. Le fenêtrage met en avant l'agencement graphique et le *libre accès* à des applications et à des documents multiples, mais avec une notion d'*exclusivité.*
– **Les widgets**[6] **:** Les *widgets* sont des objets interactifs génériques. En général regroupés dans des *barres d'outils* elles-mêmes contenues dans des fenêtres, ils comprennent les boutons, les barres de défilement, les cases à cocher, ainsi que les menus et les champs texte décrits précédemment (figure 1.1). Le paradigme des widgets met en avant les aspects d'organisation spatiale des objets d'intérêt, l'*accès direct* à ces objets, et leur *manipulation générique*.

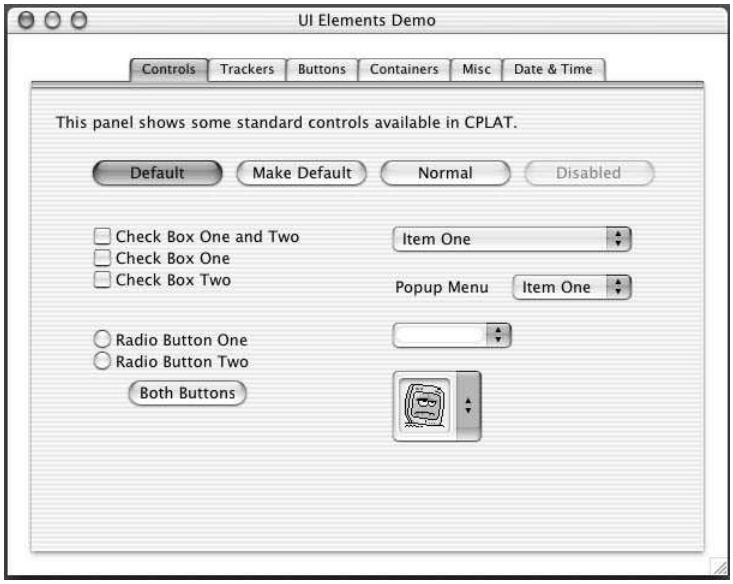


FIG. 1.1 – Quelques widgets standard des nouveaux systèmes Apple [APPLE 02].

Les interfaces actuelles s'inspirent principalement du *paradigme WIMP*, mais emploient toujours des formulaires, des systèmes de menus et plus rarement, des langages de commande. L'omniprésence des widgets provient principalement de la rapidité avec laquelle elles permettent de construire des applications interactives. À l'inverse, la manipulation directe à proprement parler (c'est-à-dire la manipulation d'objets sans widgets intermédiaires), dédiée à la tâche, est plus difficile à mettre en œuvre et n'est employée que lorsque les widgets se révèlent insuffisants (infographie, jeux). Les types de widgets existants sont en nombre limité, et leur apparence *(look)* et la façon de les contrôler *(feel)*

[6]Contraction approximative de *Window Objects*. Le terme *interactor* et en particulier sa traduction française *interacteur* sont également employés, mais désignent des concepts différents dans certains formalismes et outils, et peuvent par conséquent prêter à confusion. La confusion est plus grande encore avec les appellations *control/contrôle* ou *component/composant*. Nous leur préférerons donc le terme original de *widget*.

sont standardisés (figure 1.1 page ci-contre). Ils obéissent à un vocabulaire gestuel simple, comprenant le *pointage*, le *clic*, le *double-clic* et le *cliquer-glisser*, avec parfois l'emploi de touches modificatrices.

1.2 Les dispositifs d'entrée non standard

La partie physique de l'interface est celle avec laquelle l'utilisateur a le contact le plus direct, et les dispositifs d'entrée contribuent grandement à l'image mentale que celui-ci se fait d'un l'ordinateur [BUXTON 86B].

Depuis plusieurs années, la configuration des ordinateurs de bureau n'a guère évolué, et le clavier et la souris sont depuis longtemps devenus des dispositifs standard. Bien que cette standardisation puisse sembler souhaitable, il existe un grand nombre de situations où il est préférable, voire nécessaire, d'avoir recours à des dispositifs non standard. De nombreux travaux ont notamment montré qu'il était essentiel que les dispositifs d'entrée soient adaptés aux tâches à accomplir, ainsi qu'aux utilisateurs et à l'environnement de travail.

Cette section a pour but de donner au lecteur un aperçu de la grande variété des dispositifs d'entrée non standard, produits commerciaux ou prototypes de recherche, ainsi que les contextes dans lesquels leur emploi est pertinent. Cette présentation sera organisée autour des trois thèmes précédemment cités : tâches, utilisateurs et environnement. Pour compléter, nous évoquerons les évolutions et tendances actuelles concernant les équipements personnels.

1.2.1 Des dispositifs adaptés à la tâche

Le clavier et la souris sont des dispositifs d'entrée génériques qui permettent de communiquer avec une grande variété d'applications. Il existe néanmoins un grand nombre de tâches pour lesquelles ces dispositifs ne sont pas adaptés. Nous évoquerons deux domaines applicatifs où cette inadéquation suscite fréquemment l'emploi de dispositifs dédiés : la modélisation 3D et les jeux vidéo. Par la suite, nous verrons plus précisément en quoi un dispositif peut être adapté ou non à une tâche, et comment les exigences d'une bonne adéquation justifient l'immense variété de dispositifs d'entrée existants.

Les dispositifs dédiés dans la modélisation 3D et les jeux

Six paramètres sont nécessaires pour décrire la position et l'orientation d'un objet dans l'espace. Dans les applications d'animation ou de modélisation 3D, on dit que ces objets possèdent six degrés de liberté : trois en translation et trois en rotation. La manipulation de ces objets est fastidieuse avec un dispositif 2D comme la souris, qui ne possède que deux degrés de liberté en translation. L'une des techniques consiste à utiliser séquentiellement plusieurs modes de manipulation, dans lesquels on contrôle deux dimensions de l'objet à la fois.

Il existe sur le marché des *dispositifs à six degrés de liberté* qui sont couramment utilisés pour les tâches de manipulation 3D (figure 1.2, image de gauche) : moins maniables qu'un pointeur classique, ils se révèlent toutefois bien plus efficaces pour le positionnement rapide d'un objet dans l'espace. Pour des tâches de positionnement plus spécifiques, la mise en place d'une solution adaptée peut se révéler encore plus intéressante. La figure 1.2 présente au centre une technique de manipulation

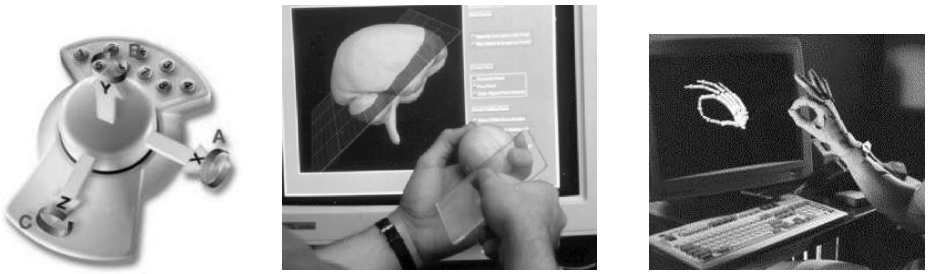


FIG. 1.2 – Les six degrés de liberté d'un dispositif 3D *Magellan* (à gauche), deux objets physiques munis d'un capteur de position dans une application de planification neuro-chirurgicale (au centre), et un gant de données *Cyberglove* d'Immersion 3D possédant 22 capteurs articulaires (à droite).

d'un plan de coupe dans une application de planification neurochirurgicale : des *capteurs positionnels 3D*[7] ont été placés sur deux objets physiques, le premier figurant le plan de coupe et le second l'objet à couper [HINCKLEY *et al.* 94A]. Enfin, dispositif classique dans le domaine de la 3D, le *gant de données* possède des capteurs angulaires sur les articulations, ce qui le rend particulièrement adapté à l'animation de mains virtuelles (figure 1.2, image de droite). Le domaine de la 3D comprend de nombreux autres dispositifs dédiés, commercialisés ou non, que nous ne pouvons tous énumérer ici [HINCKLEY *et al.* 94B, ZHAI 98].

FIG. 1.3 – À gauche, la borne d'arcade du jeu *Out Run* de Sega. À droite, une lance à incendie comme dispositif d'entrée dans le jeu *Brave Fire Fighters* de Sega.

Le domaine du jeu vidéo fait également un usage intensif de dispositifs dédiés, en particulier dans les salles de jeu. Les dispositifs conçus pour les simulations de pilotage s'inspirent des tableaux de bord réels de voiture ou d'avion, et sont bien plus appropriés que les contrôleurs de jeu génériques (figure 1.3, image de gauche). Ces dispositifs dédiés permettent un contrôle plus efficace du jeu, même si la plupart s'attachent avant toute chose à reproduire des situations réelles, afin d'ajouter au réalisme et augmenter l'immersion du joueur. C'est le cas par exemple des simulations de sport de glisse utilisant comme entrée les impulsions du joueur sur une planche, ou des jeux de « tir » utilisant des reproductions d'armes à feu, voire des dispositifs de pointage aussi peu maniables que des lances à incendies (figure 1.3, image de droite).

[7] Ces capteurs transmettent la position et l'orientation d'un objet physique dans l'espace.

Origines d'une grande diversité : les exigences de la compatibilité

Le marché propose une grande variété de dispositifs d'entrée pour la plupart inconnus du grand public, et dans laquelle un concepteur d'applications dédiées a la possibilité de puiser pour rendre l'interaction bien plus efficace et naturelle. Cette grande diversité n'est pas fortuite, et une erreur courante consiste à croire qu'il suffit de choisir le « meilleur dispositif » parmi ceux qui « font la même chose ». Dans ses travaux, William Buxton met régulièrement en évidence la difficulté d'organiser les dispositifs en classes d'équivalence, et de comparer leurs performances de façon absolue. Selon lui, *chaque dispositif possède des caractéristiques propres qui le rendent extrêmement efficace pour un petit nombre de tâches, et peu approprié pour les autres* [BUXTON 86B].

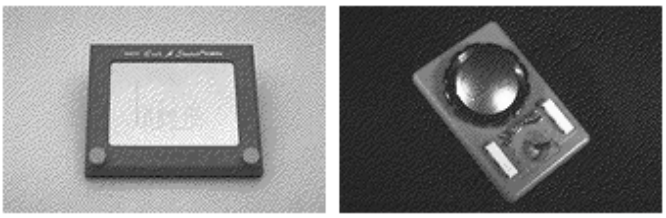


FIG. 1.4 – Deux types de Télécran : le *Etch-a-Sketch* (à gauche) et le *Skedoodle* (à droite) [BUXTON 86B].

Pour illustrer ses propos, Buxton prend pour exemple deux variantes du jouet *Télécran* (figure 1.4). La première comporte deux boutons séparés pour le dessin, que la deuxième a intégrés dans une simple manette. Le second système pourrait être considéré à raison comme une amélioration du premier, car le dessin y est plus aisé. Cependant, le premier système est bien plus efficace pour le dessin de formes géométriques à base de traits horizontaux ou verticaux (figure 1.5).

FIG. 1.5 – Deux tâches de dessin : une figure géométrique (à gauche) et une écriture cursive (à droite) [BUXTON 86B].

Cet exemple montre la relative subtilité de la notion de *compatibilité* entre un dispositif et une tâche. La *compatibilité stimulus/réponse*, c'est-à-dire le degré de similitude entre les actions physiques effectuées sur le dispositif et les effets qu'elles produisent sur l'application, constitue une dimension importante de cette compatibilité. Une autre dimension importante est la compatiblité des *propriétés intrinsèques* du dispositif, en particulier *mécaniques*, avec les caractéristiques de la tâche. Ces notions seront illustrées dans un autre exemple imaginé par Buxton.

La figure 1.6 page suivante représente deux dispositifs possédant chacun deux dimensions de translation et une dimension de rotation : une manette avec un manche rotatif monté sur ressort, et un trackball[8] sensible à la rotation d'axe vertical. Ces deux dispositifs, techniquement compatibles, ont

[8]Un trackball est une sorte de souris retournée. Des mouvements répétés de la main font rouler la boule, qui déplace le curseur à l'écran.

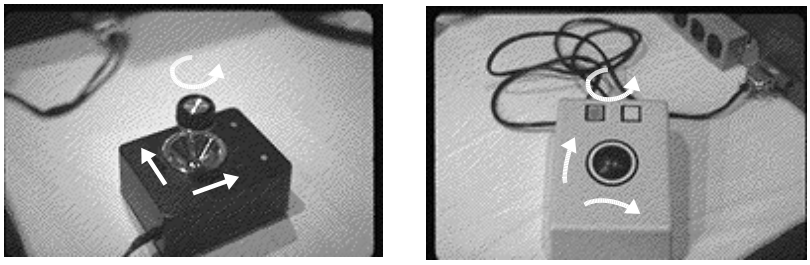


FIG. 1.6 – Une manette 3D (à gauche) et un trackball 3D (à droite) [BUXTON 86B].

pourtant des applications différentes. Le trackball permet de déplacer un objet à l'écran de façon très naturelle, car les mouvements effectués par la main sont très fortement liés aux déplacements qu'elles produisent sur l'objet. Sur la manette, la position du manche joue sur la *vitesse* et non sur la *position* de l'objet contrôlé[9], d'où une plus faible *compatibilité stimulus/réponse*, et un contrôle moins aisé. Mais contrairement à la manette, le trackball ne permet pas l'utilisation simultanée de la translation et de la rotation. Par conséquent la manette, de par ses *propriétés intrinsèques*, serait un meilleur choix pour la navigation dans une interface de type cartographique, où il est intéressant de pouvoir se déplacer et zoomer tout à la fois.

Imaginons enfin une interface de contrôle dans une raffinerie de pétrole, où une partie du système est représentée par des tuyaux et des robinets. Le contrôleur peut s'il le juge nécessaire, pointer sur un robinet avec la manette, puis le tourner dans le sens désiré. L'utilisation d'une manette sera sujette à des erreurs de manipulation, car il est difficile de tourner le manche sans agir sur les autres dimensions. Le trackball est à l'inverse plus adapté à ce type de tâche, de par ses propriétés intrinsèques.

1.2.2 Des dispositifs adaptés à l'utilisateur

Certains dispositifs spécialisés ont été conçus pour des utilisateurs qui possèdent un savoir-faire particulier, et leur permettent d'être beaucoup plus efficaces en reproduisant des gestes qu'ils maîtrisent. D'autres dispositifs ouvrent l'accès à l'informatique aux utilisateurs pour qui la manipulation d'une souris ou d'un clavier n'est pas possible.

Le transfert d'expertise

Malgré l'informatisation rapide du monde du travail, il existe encore des activités pour lesquelles les bénéfices d'une informatisation « classique » ne justifieraient pas le coût qu'imposerait la remise en cause des méthodes de travail traditionnelles. C'est dans ces situations que les démarches de conception *centrées sur l'utilisateur* [NORMAN & DRAPER 86, MACKAY *et al.* 98] prennent toute leur importance, car elles permettent de maximiser le transfert des savoirs-faire existants sur les systèmes informatisés. Nous verrons par deux exemples que le choix des dispositifs d'entrée peut prendre une part importante dans cette démarche.

[9] Le choix de ce type de contrôle est lié quant à lui aux propriétés intrinsèques du dispositif : on pourrait choisir de lier la position du manche à la position de l'objet, mais la faible résolution du dispositif et surtout le recentrage automatique du manche rendrait la tâche extrêmement difficile.

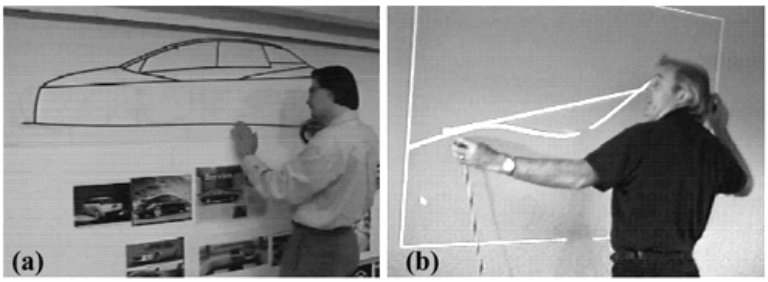


FIG. 1.7 – À gauche, un artiste dessinant avec la méthode traditionnelle de *Tape Drawing*. À droite, le *Digital Tape Drawing*, simulant le comportement physique d'une bande adhésive [BALAKRISHNAN *et al.* 99].

Dans l'industrie automobile, les concepteurs travaillant sur les silhouettes des nouveaux véhicules utilisent une technique appelée *Tape Drawing*, où les lignes principales de l'automobile sont dessinées à échelle réelle en déroulant un ruban adhésif noir sur une surface verticale (figure 1.7, image de gauche). Cette technique présente beaucoup d'avantages, dont celui de pouvoir dessiner aussi bien des lignes droites que des courbes parfaites. Ravin Balakrishnan a conçu un système de *Digital Tape Drawing* après avoir observé les méthodes de travail de ces artistes [BALAKRISHNAN *et al.* 99]. L'interface utilise deux capteurs positionnels 3D comme dispositifs d'entrées (un dans chaque main) et permet de reproduire une grande partie des techniques utilisées par les experts du *Tape Drawing* (figure 1.7, image de droite).

FIG. 1.8 – L'armature mécanique *Monkey* de Digital Image Design. Les parties peuvent être réassemblées pour former d'autres personnages [KNEP *et al.* 95].

Dans les films en images de synthèse, les animations sont effectuées sur des logiciels spécialisés qui réclament un apprentissage important. Le domaine est encore jeune, et les animateurs très expérimentés sont rares. Pour certaines scènes délicates du film *Jurassic Park*, des dispositifs spéciaux ont été conçus afin d'employer l'expérience des animateurs « image par image » traditionnels, dont le travail habituel est de faire prendre des poses successives à des pantins articulés appelés *armatures* [KNEP *et al.* 95]. Le dispositif d'entrée, surnommé *Dinosaur Input Device*, est une armature de dinosaure enrichie de capteurs angulaires sur les articulations (jusqu'à 74 au total), et dont les mouvements sont retransmis au modèle 3D. Le principe a été réutilisé avec succès dans d'autres productions,

et des dispositifs analogues sont désormais disponibles dans le commerce, comme l'armature modulaire *Monkey* (figure 1.8 page précédente).

L'accessibilité aux handicapés

La précepte selon lequel l'informatique doit être accessible à tous guide aujourd'hui la plupart des plans nationaux sur les nouvelles technologies de l'information [GSA 91]. Alors que la plupart des utilisateurs souffrant de handicaps légers peuvent employer les dispositifs standard moyennant quelques adaptations logicielles ou matérielles, des équipements spécialisés deviennent indispensables dans les cas de handicaps plus sévères. Grâce aux avancées technologiques, ceux-ci sont de plus en plus nombreux et efficaces.

L'objectif des dispositifs d'entrée dits *d'accessibilité* est de fournir une alternative au clavier et/ou à la souris lorsqu'ils sont inutilisables. L'actuelle diversité des dispositifs existant traduit à peine la grande variété des handicaps, qu'ils soient visuels, auditifs, moteurs ou cognitifs. Dans la plupart des cas, une solution est spécifiquement étudiée pour un utilisateur donné. Nous décrivons dans cette section quelques techniques couramment employées.

Les alternatives au clavier. Il existe dans le commerce une grande variété de *claviers « améliorés »* : certains possèdent par exemple des grandes touches qui peuvent être pressées par un bâton placé dans la bouche. Les utilisateurs à la fois sourds et non-voyants emploient un clavier braille en combinaison avec un affichage braille. La *reconnaissance vocale* reste cependant l'alternative au clavier la plus courante [PIEPER & KOBSA 99]. D'autres techniques ont été explorées, telles que la lecture automatique des mouvements de lèvres [STORK & HENNECKE 96] ou celle de la langue des signes [STARNER & PENTLAND 95].

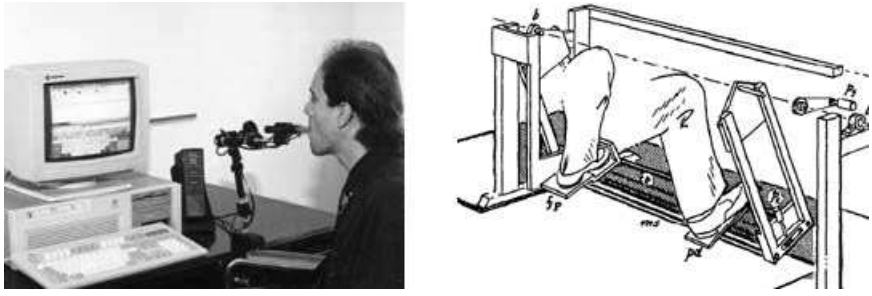


FIG. 1.9 – À gauche : un dispositif de pointage contrôlé avec la bouche. À droite : un prototype de souris à pied, le *Swing Maule* [PEARSON & WEISER 86].

Les pointeurs alternatifs. D'autres parties du corps peuvent remplacer les mains dans les tâches de pointage. Une grande variété de dispositifs, allant des manettes spécialisées à des techniques de traitement d'image, permettent aux utilisateurs tétraplégiques de déplacer un curseur par des mouvements de la tête, de la bouche ou des yeux. Un dispositif contrôlé à la langue [SALEM & ZHAI 97] a notamment permis d'atteindre des performances seulement de 5 à 50% inférieures au contrôle manuel[10] (figure 1.9, image de gauche). Il existe également un certain nombre de souris contrôlées au pied, baptisées *taupes*, dont l'un des premiers prototypes est reproduit sur la figure 1.9, image de droite

[10] La langue peut être contrôlée plus finement que la plupart des autres muscles. Voir à ce propos la figure 1.21 page 20.

[PEARSON & WEISER 86]. Les pointeurs alternatifs sont également conseillés aux utilisateurs souffrant de douleurs tensionnaires répétitives liées à l'utilisation de l'ordinateur.

Le simple bouton. Certaines personnes à mobilité extrêmement réduite sont uniquement capables de déplacer, de façon limitée, une partie de leur corps : plier un doigt, cligner de l'œil, ou presser une pédale. Ces muscles peuvent agir comme un simple bouton, dont l'état est soit ouvert soit fermé. Les ondes cérébrales peuvent également être exploitées comme modalité d'entrée binaire [DOHERTY *et al.* 01]. Si la bande passante de ces dispositifs est extrêmement faible, l'interaction est possible avec des techniques de type *Oui/Non* [DOHERTY *et al.* 01] ou des méthodes de *scanning*, où des objets (mots ou lettres de l'alphabet, par exemple) sont successivement activés à l'écran et choisis par l'utilisateur au moment voulu [STERIADIS & CONSTANTINOU 03]. Des techniques comme le *radar mouse* ou le *crosshair mouse* permettent notamment d'émuler le comportement d'une souris [WORDSPLUS 03]. Lorsque les choix possibles sont nombreux, des techniques de sélection dichotomique permettent d'accélérer le processus.

1.2.3 Des dispositifs adaptés à l'environnement

Les contextes d'utilisation de l'informatique ne se limitent pas aux environnements de type bureau. Il existe un grand nombre de situations où l'emploi d'une souris ou d'un clavier est inadapté, voire impossible.

Bornes interactives et informatique embarquée

FIG. 1.10 – À gauche : une borne d'information utilisant un écran tactile. À droite : la "molette" employée dans le Distributeur de Billets Régionaux de la SNCF.

Les *bornes interactives* sont des appareils déployés dans des lieux publics afin de fournir un libre accès à des informations ou à un service. Ces appareils doivent répondre à des contraintes importantes liées à une utilisation en lieu public : accès immédiat adapté à un contexte "libre-service" (utilisation en position debout par exemple), simplicité et accessibilité aux personnes non habituées à l'informatique, et robustesse (en particulier dans les lieux exposés au vandalisme et aux intempéries) [VAN KAMPEN 00]. L'*écran tactile*, qui répond à ces trois critères, est le dispositif le plus couramment utilisé dans les applications d'information (figure 1.10, image de gauche). Les distributeurs automa-

tiques emploient des solutions plus spécifiques, tels que le pavé numérique pour les automates bancaires, ou les pointeurs monodimensionnels dans les distributeurs de titres de transport (figure 1.10 page précédente, image de droite).

L'*informatique embarquée* désigne des appareils informatiques qui sont intégrés à des objets électroniques ou mécaniques, tels que les appareils hi-fi et électroménagers, les véhicules automobiles ou les avions[11]. Ce domaine met l'accent sur la miniaturisation et la fiabilité, mais également sur les qualités ergonomiques. Les *panneaux de contrôle* et les *tableaux de bord* constituent des exemples typiques de dispositifs d'entrée dédiés à la tâche, aussi bien par leur conception que par leur agencement spatial. En informatique embarquée, il est courant de devoir prendre en compte les spécificités de l'environnement : dans une voiture par exemple, le bruit ambiant est important et les mains et les pieds sont constamment occupés.

Informatique mobile et vestimentaire

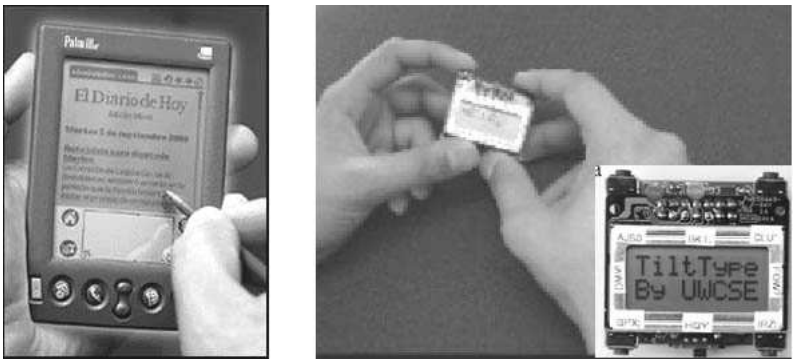


FIG. 1.11 – À gauche : un assistant personnel. À droite : la technique *TiltType*, qui permet de saisir du texte sur des appareils de très petite taille équipés de quatre boutons et d'un accéléromètre qui distingue huit inclinaisons possibles [PARTRIDGE *et al.* 02].

L'informatique mobile ou *nomade* désigne des outils informatiques légers que l'on conserve sur soi lors de ses déplacements. Elle nécessite des dispositifs d'entrée compacts et compatibles avec une activité nomade. Ainsi, la plupart des *ordinateurs de poche* comme les assistants personnels comportent un écran sensitif et la souris et le clavier sont remplacés par un simple stylet (figure 1.11, image de gauche). En milieu professionnel, un installateur téléphonique peut par exemple employer un ordinateur de poche muni d'un système de communication radio pour saisir et recevoir des commandes sur son lieu de travail [PREECE *et al.* 94]. La reconnaissance vocale est également employée dans l'informatique mobile, par exemple pour la composition de numéros sur des téléphones portables. D'autres techniques d'interaction ont été récemment proposées pour les appareils mobiles : l'une de celles-ci permet d'utiliser un ordinateur de poche avec une seule main en agissant simplement sur son inclinaison [REKIMOTO 96]. Cette méthode a été ensuite adaptée à la saisie textuelle sur des appareils de très petite taille [PARTRIDGE *et al.* 02] (figure 1.11, image de droite).

Les *ordinateurs vestimentaires* constituent un domaine particulier de l'informatique mobile. Portés comme des accessoires vestimentaires, ces ordinateurs permettent d'accomplir des tâches informa-

[11] Habituellement, ce terme inclut également l'*informatique mobile*, que nous réservons pour la section suivante.

FIG. 1.12 – Évolution au fil des années de l'équipement de Steve Mann, l'un des *cyborgs* du MIT qui portent en permanence des ordinateurs vestimentaires dans de but de les tester et de les améliorer [MANN 96].

tiques courantes ou spécifiques sans interrompre son activité, l'exemple classique consistant à rédiger un courrier électronique tout en se promenant dans un lieu public. De tels systèmes emploient traditionnellement des dispositifs visuels ou sonores non intrusifs associés à des dispositifs d'entrée variés, allant de simples boutons placés à des endroits stratégiques, à des systèmes à base de reconnaissance vocale. Les *chord keyboards*, claviers légers comportant un nombre restreint de touches employées de façon combinée, sont couramment utilisés [JACOB 96]. Des informations en provenance de l'environnement peuvent également être exploitées (GPS, capteurs biométriques). L'informatique peut également être intégrée aux vêtements grâce au *textile intelligent* qui est, entre autres, sensible au toucher [MANN 96].

1.2.4 L'évolution de l'équipement grand public

L'évolution de l'équipement grand public est lente mais réelle. Après plusieurs décennies de stagnation, nous assistons actuellement à deux tendances importantes : une *complexification* des dispositifs standard d'une part, et une *démocratisation* de certains autres périphériques d'autre part.

Complexification des périphériques standard

Malgré la grande inertie du marché des dispositifs d'entrée grand public, il n'est pas rare de voir les dispositifs standard subir des extensions significatives. L'ajout sur les souris d'une molette servant à faire défiler les documents, pourtant récente, est devenue un quasi-standard sur les systèmes Windows.

Aujourd'hui, l'ajout de contrôles additionnels sur les périphériques standard est de plus en plus courant. La plupart des souris modernes possèdent un ou deux boutons latéraux supplémentaires, faisant usage du pouce et de l'annulaire (Figure 1.13 page suivante, image de gauche). Ces boutons peuvent être assignés à des opérations courantes telles que la navigation dans l'historique des pages Internet. Les claviers « Internet » proposent également des boutons de navigation, ainsi que des contrôles dédiés à l'écoute de CDs audio (figure 1.13 page suivante, au centre). Enfin, la plupart des manettes modernes possèdent un ou deux contrôles analogiques supplémentaires : une *manette des*

FIG. 1.13 – Trois exemples de périphériques récents : à gauche, la souris *Internet Navigator* de Logitech et ses cinq boutons, configurés pour le jeu *Half-Life*. Au centre, les contrôles audio du clavier *Intellimouse Explorer* de Logitech. À droite, une partie des contrôles de la manette de jeu *Cougar HOTAS* de Thrustmaster.

gaz et un *chapeau multidirectionnel* servant à spécifier la direction du regard, et il n'est pas rare de voir jusqu'à 14 boutons disposés sur l'ensemble du dispositif (figure 1.13, image de droite).

La complexification des dispositifs de saisie est une conséquence naturelle de la diversification et de la complexification des tâches. Dans l'industrie du jeu, la multiplication considérable des contrôles sur les manettes de jeu modernes s'explique par des simulations de plus en plus réalistes et hautement interactives qui génèrent une demande accrue quant aux capacités des dispositifs de contrôle. Sur les claviers et souris, les extensions physiques proposées par le marché ne sont pas toutes essentielles et pertinentes, et il y a fort à parier que toutes ne s'imposeront pas comme la molette. Cependant, l'ajout d'un contrôle physique peut se révéler fort profitable, en particulier lorsqu'il est assigné à une tâche répétitive. La plupart des boutons sont programmables, et contrairement aux touches de fonction du clavier, sont souvent aisés à discerner et à mémoriser, et possèdent un comportement similaire entre les applications.

Démocratisation de périphériques non standard

FIG. 1.14 – Trois dispositifs grand public à prix abordable (microphone, webcam et tablette graphique), ainsi qu'un ordinateur personnel configuré pour un jeu de course automobile.

Jusque là, seuls les utilisateurs avancés (les joueurs passionnés, par exemple) étaient prêts à investir dans des équipements spécialisés coûteux. Maintenant, grâce à la baisse des coûts du matériel et en particulier de certains périphériques, le grand public peut progressivement se rendre compte de l'intérêt que peuvent présenter les dispositifs d'entrée non standard. Des périphériques comme le

microphone, la webcam (que certains jeux spécialisés exploitent [SONY 03]) et la tablette graphique (livrée gratuitement avec certains logiciels) sont maintenant de plus en plus courants dans les configurations matérielles[12] (figure 1.14 page ci-contre). Il y a fort à parier que la configuration rigide de l'ordinateur de bureau évolue à terme vers des configurations de plus en plus riches et hétérogènes en termes d'entrée. À condition, bien sûr, que les applications apprennent à les exploiter au mieux.

1.3 Nouveaux paradigmes d'interaction

De nouvelles techniques d'interaction sont régulièrement proposées dans la littérature scientifique dans le but de rendre l'interaction plus efficace et plus naturelle. La plupart de ces publications montrent que même l'interaction avec des applications conventionnelles employant des dispositifs standard est susceptible d'être sensiblement améliorée.

Le terme d'*interface post-WIMP* est couramment évoqué pour décrire des interfaces employant ces nouveaux paradigmes d'interaction. Bien qu'il n'existe pas de définition consensuelle de ces interfaces, nous pouvons en dégager quelques fils conducteurs. Nous les décrivons ici, avant de passer en revue des exemples spécifiques de nouveaux paradigmes d'interaction, à savoir : l'interaction gestuelle, les outils transparents, l'interaction parallèle et la réalité mixte.

1.3.1 Objectifs et enjeux des interfaces post-WIMP

Les nouveaux paradigmes d'interaction tendent essentiellement vers des interactions toujours plus *concises* et plus *directes*.

La notion de concision est en rapport étroit avec le concept de *phrasé* decrit par Buxton [BUXTON 86A]. Dans une interface conventionnelle, l'ouverture d'un menu suivi du choix d'un de ses éléments peut se faire en deux clics. Ces clics successifs forment en quelque sorte une *phrase*. Or selon Buxton, toute phrase traduit une connectivité conceptuelle entre les tâches atomiques, qu'il est préférable de renforcer par une tension physique. Autrement dit, *tout concept ou transaction pouvant être décrit en un seul mot ou une seule phrase devrait pouvoir être articulé en un seul geste* [BUXTON 86A]. Lorsqu'un menu est ouvert par la méthode du cliquer-glisser, la tension musculaire est maintenue entre l'ouverture du menu et le choix d'un élément, ce qui rend impossible toute erreur de syntaxe. Le geste est en outre plus fluide et plus naturel.

La propriété de concision est également liée à la propriété d'*amodalité*. Les *modes* dans les interfaces désignent des états dans lesquels les entrées sont interprétées différemment [POLLER & GARTER 84] : par exemple, les modes « insertion » et « remplacement » dans un traitement de texte. Les modes peuvent générer des ambiguïtés et être à la source d'erreurs appelées *erreurs de mode*[13] [NORMAN 81]. En outre, les changements de modes sont autant d'étapes supplémentaires dans l'interaction, et peuvent la ralentir considérablement (palette d'outils). Les nouvelles techniques d'interaction visent à supprimer ces changements de mode, ou à les rendre plus rapides et naturels (*quasi-modes*).

Tout le monde s'accorde également à dire que l'interaction doit être la plus *directe* possible, bien

[12] À titre d'exemple, les ventes de webcam aux États-Unis ont grimpé de 36% de 2001 à 2002 [GILL 02].

[13] Les erreurs de mode ont été à l'origine de nombreuses catastrophes. Le crash de l'Airbus A320 sur le Mont-Saint-Odile en 1992 en est un exemple typique : peu avant l'atterrissage, le pilote, très occupé, a saisi une vitesse de descente de 3300 pieds/mn au lieu d'un angle de 3.3 degrés [HOURIZI & JOHNSON 01]

que cette notion reste encore relativement vague.

Le principe de la *manipulation directe* [SHNEIDERMAN 83] a exercé une forte influence sur le développement des interfaces grand public. En pratique cependant, l'interaction est loin d'être toujours directe : les boîtes de dialogues et les widgets, en particulier, occupent de l'espace sur l'écran et déplacent l'attention de l'utilisateur hors des objets d'intérêt [BEAUDOUIN-LAFON 97, BEAUDOUIN-LAFON 00]. Le modèle de l'*interaction instrumentale* (voir section 2.5.2 page 57) emploie le terme de *degré d'indirection* pour désigner à la fois le décalage temporel (modes) et le décalage spatial suscité par ces techniques. De nouvelles techniques sont régulièrement proposées pour revenir à une interaction plus locale, essentiellement centrée sur les objets d'intérêt.

La *compatibilité stimulus/réponse* [BUXTON 86B, NORMAN 02] que nous avons évoquée dans la section 1.2.1 page 7 met en avant la compatibilité entre le dispositif d'entrée physique et la tâche. Jacob et al. [JACOB *et al.* 94] ont par ailleurs confirmé que les performances sont faibles lorsque que des *tâches intégrales* (comme le placement d'un objet 3D) sont morcelées par l'emploi d'un dispositif dont le nombre de degrés de libertés est insuffisant. L'ensemble de ces résultats incite à une *simplification des techniques d'interaction* à travers l'usage de dispositifs dédiés. Dans les procédés de contrôle direct caractérisés par la quasi-absence de techniques d'interaction, la sensation de contrôle et d'engagement se trouvent intensifiés. Ces dimensions sont notamment dominantes dans les jeux vidéo.

Tout comme l'interaction concise, l'interaction directe suppose une *continuité* plutôt qu'une *séquentialité* dans les actions.

1.3.2 L'interaction gestuelle

L'interaction gestuelle constitue un moyen non conventionnel d'employer les dispositifs de pointage en exploitant notre capacité musculaire à mémoriser et à reproduire des trajectoires, capacité que nous employons notamment pour écrire.

Exploiter la dynamique du mouvement

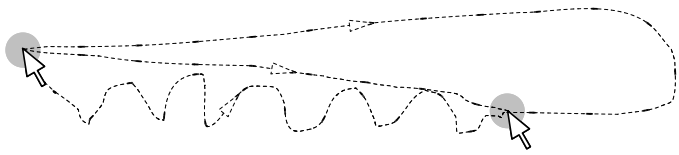


FIG. 1.15 – Dans les interfaces classiques, lorsqu'un utilisateur effectue une action de type cliquer-glisser, seules la position initiale et la position finale du pointeur sont pris en compte, indépendamment de la trajectoire effectuée.

Dans la plupart des interfaces à manipulation directe, les dispositifs de pointage servent essentiellement à spécifier des *positions*[14]. Dans un mouvement de cliquer-glisser par exemple, seules la position initiale et la position finale sont interprétées (figure 1.15). En exploitant la *dynamique* du mouvement, les techniques d'interaction gestuelles permettent d'enrichir la sémantique des actions élémentaires de l'utilisateur [BAUDEL 95].

[14]Les *menus hiérarchiques* constituent une exception [MERTZ *et al.* 00], pouvant tout au plus être considérée comme un

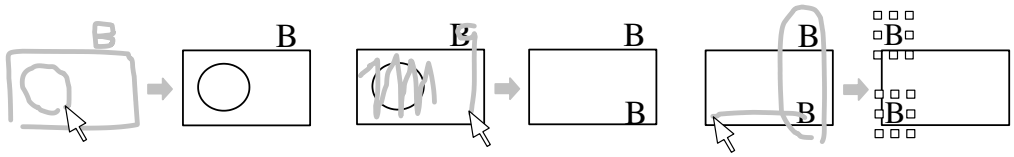


FIG. 1.16 – Exemples usuels de techniques d'interaction gestuelle dans une application graphique : l'utilisateur commence par créer un rectangle, un cercle, et un label. Il supprime ensuite le cercle et duplique le label. Enfin, il regroupe les deux labels et les déplace vers la gauche.

La figure 1.16 illustre quelques techniques d'interaction gestuelle classiques, telles qu'on peut les trouver dans des applications comme [MANKOFF *et al.* 00] ou [HONG & LANDAY 00]. Chaque action de type cliquer-clisser produit, à la manière d'un logiciel de dessin, une *trace*, qui disparaît une fois interprétée. La trace pourra être interprétée comme un nouvel objet à créer, dont le type la taille et la position seront déduits des caractéristiques du geste. Elle pourra également être interprétée comme une commande d'édition, dont la nature et les objets ciblés seront également déduits du geste selon un algorithme prédéterminé.

Interfaces gestuelles : des avantages et des inconvénients

L'avantage principal de l'interaction gestuelle est qu'elle permet d'effectuer des interactions *concises* [BAUDEL 95]. Ainsi, la plupart des opérations gestuelles de l'exemple précédent réclament habituellement au moins deux étapes dans une interface classique : créer un rectangle nécessite de cliquer sur l'outil « rectangle » puis de désigner deux côtés opposés du rectangle par un cliquer-glisser. De même, la suppression d'un objet requiert une sélection, puis l'appui d'une touche ou un clic sur le bouton « supprimer ». Or, un geste suffit pour spécifier à la fois une commande et les arguments de cette commande. En outre, l'interaction gestuelle n'utilise pas de widgets intermédiaires tels que les barres d'outils, ce qui économise de la place sur l'écran et permet à l'utilisateur de se focaliser sur les objets d'intérêt.

L'autre atout de l'interaction gestuelle est qu'elle exploite une métaphore déjà connue, à savoir le dessin sur une feuille de papier. L'utilisation de stylets et l'exploitation de gestes iconiques tel que le geste de rature (figure 1.16) exploitent davantage encore la puissance de cette analogie. Ainsi, les techniques d'interaction gestuelle ont été largement utilisées dans les applications conçues selon une approche d'« interaction informelle »[15] et abondamment exploitées pour la saisie textuelle [LANDAY & MYERS 95]. L'interaction gestuelle est également la technique de prédilection des ordinateurs de poche, qui utilisent le stylet comme principal dispositif de saisie, et pour lesquels il convient de maximiser l'espace d'affichage utile.

Bien que fort prometteuses, les interfaces essentiellement basées sur les techniques gestuelles restent difficiles à mettre en œuvre. Les ambiguïtés potentielles, par exemple, sont nombreuses : ainsi, dans l'exemple de la figure 1.16, le cercle pourrait être interprété comme la lettre « O », et le déplacement d'objets comme la création d'un segment de droite. Plus encore que les difficultés liées à

exemple primitif et très imparfait d'interaction gestuelle.

[15]Cette approche vise à ne pas faire obstacle aux capacités d'expression créative de l'utilisateur en privilégiant des méthodes de saisie naturelles et en acceptant des données incomplètes ou imprécises (telles que des croquis), tout en minimisant les interventions de l'ordinateur.

l'apprentissage, les erreurs de reconnaissance résultant d'un vocabulaire gestuel complexe sont une source de frustration considérable pour l'utilisateur. La tendance actuelle semble ainsi aller vers l'utilisation de gestes simples, plus rapides et plus tolérants aux erreurs de par leur construction logique [MACKENZIE & ZHANG 97, NG *et al.* 98], mais ce au détriment de leur facilité d'assimilation dû à leur nature arbitraire.

Les gestes ergotiques

Les gestes permettent d'explorer le monde physique (*fonction épistémique*), d'agir matériellement sur ce monde (*fonction ergotique*), et de communiquer avec les autres (*fonction sémiotique*) [CADOZ 94].

La plupart des techniques conventionnelles emploient des gestes sémiotiques. Les quelques exemples de techniques exploitant les fonctions ergotique et épistémique montrent cependant que celles-ci possèdent de nombreux avantages : ce sont des gestes simples, souvent utilisables à la souris, qui ne réclament pas d'apprentissage car *ils s'expliquent d'eux-mêmes*. Ces techniques sont la preuve que l'interaction gestuelle n'impose pas nécessairement de compromis entre facilité d'assimilation et efficacité d'utilisation.

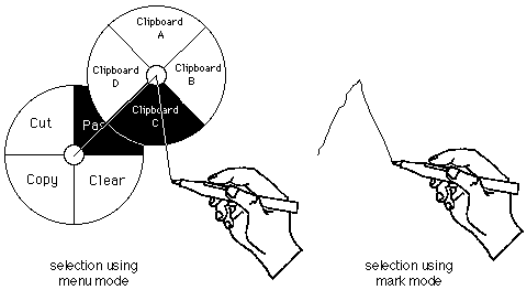


FIG. 1.17 – Sélection d'une commande dans un *Marking Menu*, par un novice (à gauche) et par un expert (à droite).

C'est le cas des *Marking Menus*, version circulaire des menus contextuels [KURTENBACH & BUXTON 94]. Dans un menu circulaire, le choix d'un élément se fait en imprimant une direction donnée au pointeur, après avoir fait apparaître le dit menu (figure 1.17). Les menus hiérarchiques des Marking Menus induisent des gestes qui sont affichés sous forme de traces (figure 1.17, image de droite). L'utilisateur novice commencera par maintenir le bouton enfoncé afin de faire apparaître les menus (figure 1.17, image de gauche), et pourra ensuite directement effectuer les gestes pour les commandes auxquelles il est habitué (figure 1.17, image de droite).

La figure 1.18 page suivante illustre un autre exemple tiré du logiciel *Digistrips*, un prototype de strips électroniques pour contrôleurs aériens [MERTZ *et al.* 00]. Les *Strips* sont des objets disposés verticalement. Un Strip peut être décalé vers la droite, ou déplacé vers le haut ou vers le bas, auquel cas il pousse les autres afin de conserver l'ordre. Mais un court mouvement vers la droite suivi d'un déplacement vertical le désolidarise des autres, ce qui lui permet d'être inséré ailleurs. Le passage à ce mode de déplacement se fait par un geste implicite, « mécanique », dont la compréhension et l'assimilation sont immédiates.

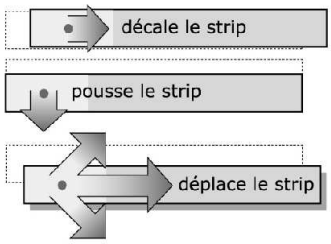


FIG. 1.18 – Les gestes reconnus pour la manipulation des Strips [MERTZ *et al.* 00].

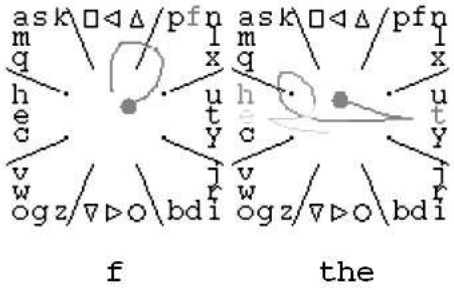


FIG. 1.19 – La lettre *'f'* et le mot *'the'* saisis avec la technique QuikWriting [PERLIN 98].

La méthode de saisie textuelle *QuikWriting* de Ken Perlin constitue un autre exemple de gestes ergotiques [PERLIN 98]. Pour la saisie textuelle au stylet sur les ordinateurs de poche, les vocabulaires gestuels simplifiés comme Graffiti [MACKENZIE & ZHANG 97] sont souvent préférées aux techniques d'« écriture naturelle » [PARAGON 03, MICROSOFT 03B] qui sont sujettes aux erreurs de reconnaissance. Cependant, ces vocabulaires gestuels nécessitent d'être appris. La technique *QuikWriting* se distingue de ces approches car elle peut être utilisée, tout comme les *Marking Menus*, de façon exploratoire. L'espace de saisie de *QuikWriting* est divisée en neuf zones (figure 1.19). Chaque geste commence dans la zone centrale, puis passe par une ou deux zones avant de revenir au centre. La correspondance entre caractères et gestes est représentée de façon synthétique sur l'espace de saisie : pour saisir une lettre, il suffit de diriger le geste dans la zone où elle se trouve, puis dans la zone indiquée par sa position relative.

1.3.3 Les outils semi-transparents

Les *outils semi-transparents*[16] sont des widgets que l'on peut librement déplacer sur l'écran, et qui sont en général regroupés dans des palettes flottantes appelées *toolglass*. Ces outils possèdent des zones transparentes qui permettent de voir les objets d'intérêt situés en-dessous[17].

Lorsqu'un utilisateur clique sur un outil semi-transparent, celui-ci modifie les propriétés de l'objet qui se trouve en dessous, à la position du curseur. Sur la figure 1.20 page suivante par exemple, un utilisateur modifie la couleur d'un objet graphique en cliquant dessus à travers un bouton transparent.

[16] en anglais, *see-through tools* ou *click-through tools*.

[17] Les *Magic Lenses* (lentilles magiques) sont une variante où la transparence est remplacée par des filtres visuels élaborés qui apportent des informations supplémentaires sur les objets.

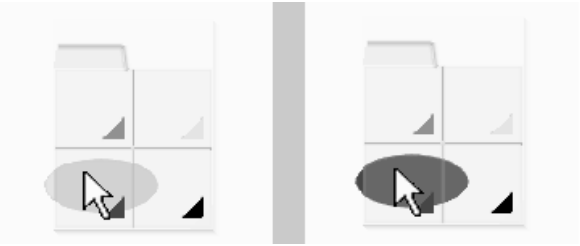


FIG. 1.20 – Une *toolglass* comportant quatre outils de changement de couleur. L'utilisateur clique sur une ellipse à travers l'un de ces outils, ce qui a pour effet de modifier la couleur de l'objet graphique.

Ce bouton permet d'affecter une couleur donnée (affichée dans le coin inférieur droit de l'outil) à tout objet graphique.

Les outils semi-transparents ont été introduits par Eric A. Bier [BIER *et al.* 93], comme alternative aux techniques indirectes de manipulation d'objets graphiques, telles que la sélection, les barres d'outils modales ou les menus contextuels. Contrairement à ces techniques, les outils semi-transparents permettent de spécifier à la fois l'objet sur lequel on veut effectuer une opération et l'opération elle-même. Couplée à l'interaction bimanuelle décrite quelques lignes plus bas (la Toolglass peut être contrôlée de la main gauche), cette technique permet d'effectuer en un seul geste des manipulations qui, avec les techniques classiques, requièrent plusieurs étapes et des déplacements répétés de l'attention visuelle.

1.3.4 L'interaction parallèle

Nous employons le terme d'*interaction parallèle* pour désigner l'ensemble les paradigmes d'interaction faisant un usage simultané de plusieurs dispositifs d'entrée.

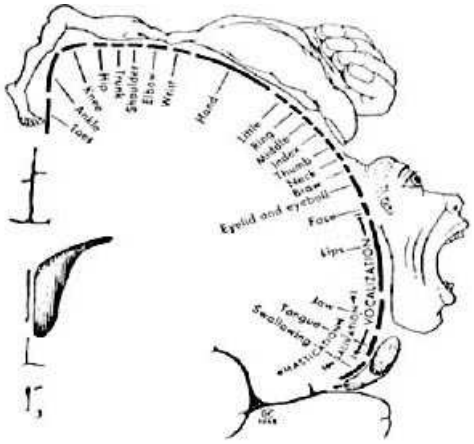


FIG. 1.21 – *Homoncule* représentant les zones du cortex moteur responsables de chaque groupe musculaire [BARLOW *et al.* 90].

L'efficacité de l'interaction peut se décrire en termes de *bande passante*, c'est-à-dire de la quantité

d'information que l'utilisateur est capable d'acheminer vers la machine en un temps donné. Actuellement cette vitesse est bien inférieure à la vitesse limite théorique, qui dépend essentiellement de la *bande passante motrice* humaine. L'*homoncule* représenté sur la figure 1.21 page ci-contre donne un aperçu de l'ensemble de cette bande passante, et montre notamment que certains groupes moteurs peuvent être contrôlés plus finement que d'autres.

L'homme est capable, par apprentissage, d'utiliser de façon conjuguée n'importe quel ensemble de groupes musculaires. C'est le cas lorsqu'il apprend à jouer d'un instrument de musique ou à conduire une voiture. En outre, nombre de gestes quotidiens instinctifs mettent en jeu ce type de coordination musculaire, et c'est également le cas pour le langage.

Dans les *interactions parallèles*, plusieurs dispositifs d'entrée sont employés simultanément afin de mieux exploiter la bande passante motrice humaine. Certains des dispositifs alternatifs que nous avons évoqués dans la section 1.2.2 page 10 mettent en jeu des groupes musculaires spécifiques et peuvent être employés en complément des dispositifs standard. Ici, nous nous contenterons de décrire les deux principaux paradigmes d'interaction reposant sur l'interaction parallèle, à savoir l'*interaction bimanuelle* et l'*interaction multimodale*.

L'interaction bimanuelle

Une expérience de William Buxton et Brad Myers [BUXTON & MYERS 86] a mis en évidence que des techniques d'interaction utilisant deux mains permettaient d'accomplir certaines tâches plus rapidement. Leur expérience consistait à spécifier d'une part la taille et la position d'une cible, et d'autre part à naviguer dans un document et sélectionner un mot. Dans les deux cas, le parallélisme était exploité de façon naturelle et augmentait significativement les performances.

Au même moment, Yves Guiard [GUIARD 87] a montré que dans les tâches manuelles courantes, les deux mains travaillent de façon coopérative et asymétrique, et a mis notamment l'accent sur l'importance de la main non-dominante. Ainsi, la main non-dominante peut servir de référentiel spatial pendant que la main dominante opère sur l'objet tenu par la main gauche, et la première effectue en général des mouvements de grande amplitude alors que la dernière effectue des mouvements précis (exemple du peintre qui tient sa palette). En outre, la main non-dominante précède souvent le geste, que la main dominante termine (prendre une feuille de papier pour écrire).

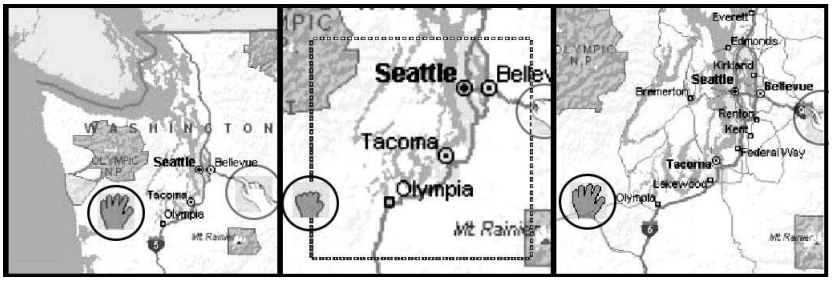


FIG. 1.22 – Navigation cartographique bimanuelle : les deux pointeurs (que nous avons entouré d'un cercle pour plus de clarté) sont utilisés simultanément pour zoomer. La carte est réactualisée à la fin de l'interaction. [HINCKLEY *et al.* 98]

L'emploi de modificateurs clavier et la saisie textuelle à deux mains constituent déjà des interac-

tions bimanuelles rudimentaires. Mais sur la base des expériences de Buxton et Myers et de la théorie de Guiard, des techniques bien plus intéressantes ont pu être explorées. Appliquées à l'infographie, de telles techniques permettent, comme le font les dessinateurs, d'orienter l'espace de travail de la main gauche tout en dessinant de la main droite [KURTENBACH *et al.* 97]. La figure 1.22 page précédente montre une autre technique bimanuelle où l'utilisation simultanée des deux mains permet de zoomer en avant ou en arrière sur une carte, tout en la déplaçant [HINCKLEY *et al.* 98].

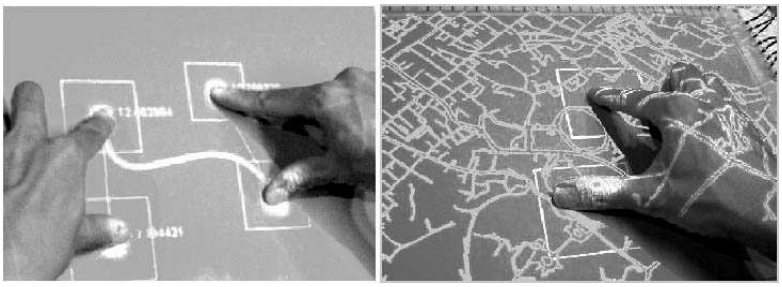


FIG. 1.23 – Manipulation « multidigitale » avec Smartskin. À gauche, la manipulation d'une courbe de Bézier. À droite, une technique de navigation cartographique similaire à celle de Hinckley [REKIMOTO 02]

D'autres techniques d'interaction bimanuelle ont été décrites dans la littérature scientifique. Nous avons évoqué trois d'entre elles dans les sections précédentes : la manipulation d'un plan de coupe (figure 1.2 page 6), le tape drawing (figure 1.7 page 9), et en particulier les outils transparents (figure 1.20 page 20). Récemment, un dispositif d'entrée expérimental mais très prometteur, nommé Smartskin, a permis d'appliquer certaines techniques bimanuelles à la manipulation « multidigitale », comme illustré sur la figure 1.23 [REKIMOTO 02].

L'interaction multimodale

Selon Laurence Nigay, la *multimodalité* est la capacité d'un système à communiquer avec un utilisateur en employant différents types de canaux de communication [NIGAY & COUTAZ 93]. En général, les interfaces multimodales emploient des entrées « naturelles » mais qui produisent des ambiguïtés comme la parole, et les complètent par des entrées explicites comme le pointage ou la manipulation directe. Bien que le terme de multimodalité soit parfois employé dans le sens plus général d'*interaction parallèle*, le domaine de l'interaction multimodale s'attache essentiellement aux problèmes liés à la fusion et à l'interprétation d'entrées issues de canaux de *nature hétérogène*.

La notion d'interfaces multimodales a été introduite par Richard Bolt [BOLT 80] dans son système « put-that-there », qui associe des commandes vocales à des techniques de pointage (figure 1.24 page suivante). Pour déplacer un objet, l'utilisateur pointe du doigt sur cet objet en prononçant « met ça », puis sur sa destination en prononçant « là ». D'autres prototypes ont été construits par la suite afin expérimenter l'usage combiné de la manipulation directe et du langage naturel textuel [COHEN *et al.* 89], ou des gestes et de la parole [WEIMER & GANAPATHY 89]. Plus récemment, des techniques de traitement vidéo en temps réel ont été exploitées pour améliorer la reconnaissance vocale par la lecture des mouvements des lèvres, et pour contrôler une application de visualisation d'images panoramiques en combinant mouvements oculaires et commandes vocales [YANG *et al.* 98].

Les interfaces multimodales offrent la possibilité de combiner les avantages des entrées naturelles

FIG. 1.24 – Le système « put-that-there » de Bolt [BOLT 80].

mais équivoques comme la parole, et des entrées univoques comme la manipulation directe. L'utilisation de plusieurs canaux de communication permet à chaque canal de compenser les faiblesses des autres, lorsqu'il s'agit notamment de résoudre des ambiguïtés, mais permet également la propriété de redondance, dans laquelle une tâche peut être accomplie de diverses manières. Cette redondance des entrées est souvent désirable, en particulier lorsque les contraintes liées à l'environnement sont variables.

1.3.5 La réalité mixte

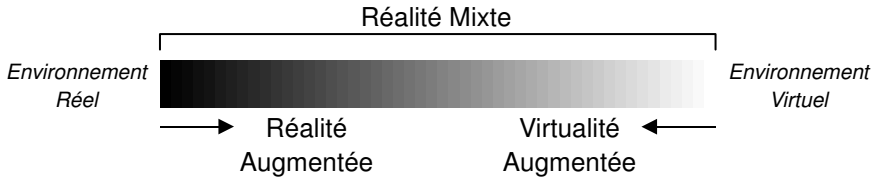


FIG. 1.25 – Le continuum réel-virtuel de Milgram.

Paul Milgram [MILGRAM & KISHINO 94] a introduit le terme de *réalité mixte* pour désigner l'ensemble des approches combinant environnements virtuels et environnements réels dans des proportions variables. Il décrit ainsi un continuum entre la réalité et la virtualité à partir duquel on peut distinguer deux grandes approches (figure 1.25) : la *virtualité augmentée* qui consiste à intégrer du réel dans le monde virtuel, et la *réalité augmentée* qui consiste à intégrer du virtuel dans le monde réel. Nous décrivons ces deux paradigmes.

La virtualité augmentée

Nous possédons des capacités innées à interagir de façon physique avec des objets du monde réel, et ces capacités sont très peu exploitées dans les interfaces actuelles où les objets sont essentiellement virtuels [FITZMAURICE & BUXTON 97, ISHII & ULLMER 97]. La *virtualité augmentée* traduit une nouvelle

tendance dans la recherche en IHM qui vise à introduire des objets physiques dans les interfaces utilisateur conventionnelles.

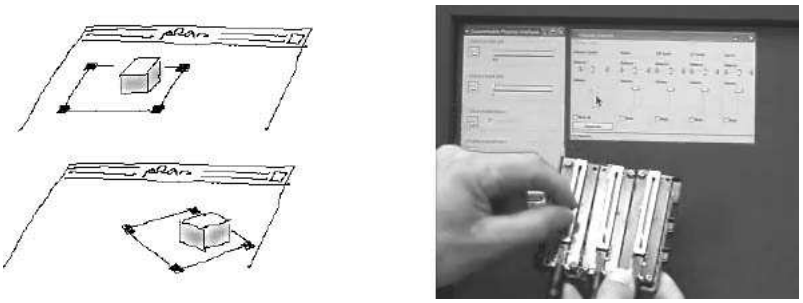


FIG. 1.26 – À gauche : *interface « saisissable »*, où des briques physiques sont utilisées pour manipuler des objets virtuels [FITZMAURICE *et al.* 95]. À droite : un assemblage de *phidgets* est assigné à des widgets de contrôle sonore [GREENBERG & BOYLE 02].

George Fitzmaurice et Hiroshi Ishii [FITZMAURICE *et al.* 95] ont employé le terme d'*interfaces « saisissables »*[18], pour décrire des interfaces où certains objets interactifs traditionnels sont remplacés par des objets physiques. Ils évoquent un prototype où des « briques » comportant des capteurs de position et d'orientation sont librement placés sur une table, sur laquelle sont projetés des objets graphiques. Lorsqu'une brique est posée sur un objet virtuel, celui-ci peut être librement manipulé (figure 1.26, image de gauche). Plusieurs briques peuvent être manipulées simultanément, soit sur des objets séparés, soit sur le même objet pour le redimensionner ou le déformer. Hiroshi Ishii [ISHII & ULLMER 97] généralisera ces concepts dans une perspective plus générale de réalité mixte, et introduira le terme maintenant couramment employé d'*interfaces tangibles*.

Dans le même esprit, Saul Greenberg [GREENBERG & BOYLE 02] propose de s'aider de contrôleurs physiques supplémentaires pour interagir avec les applications existantes. Ses *phidgets* sont les homologues physiques des widgets qui structurent nos interfaces. Ainsi, des boutons ou des potentiomètres électroniques peuvent être assignés aux fonctions les plus couramment utilisés, puis agencés sur l'espace de travail. Sur la figure 1.26, image de droite, une combinaison de potentiomètres linéaires est assignée à des fonctions de contrôle sonore.

Dans ces interfaces, les objets physiques sont plus efficaces et plus faciles d'utilisation car ils mettent à profit nos capacités préhensiles innées [FITZMAURICE & BUXTON 97]. En outre, ils permettent et même encouragent l'interaction bimanuelle et le travail coopératif.

La réalité augmentée

La *réalité augmentée* [MACKAY 96] désigne des techniques consistant à superposer au monde réel des informations générées par ordinateur. L'exemple typique est un utilisateur (chirurgien, militaire,...) qui est assisté dans sa tâche par un système *vestimentaire* (voir section 1.2.3 page 12) comprenant un casque virtuel ou des lunettes translucides. Une autre technique courante consiste à projeter des images sur un espace de travail : les *surfaces augmentées* telles que les tables, les tableaux ou les murs interactifs emploient des techniques d'interaction basées sur la capture des mouvements de la main ou s'appuient sur des interfaces tangibles [ISHII & ULLMER 97]. L'*argile lumineuse* [PIPER *et al.* 02] par

[18] Traduction approximative de *Graspable User Interfaces*.

FIG. 1.27 – À gauche : l'*argile lumineuse* avec le calcul des courbes de gradient [PIPER *et al.* 02]. À droite : une dalle de *sol intelligent* et un des capteurs de charge qui la composent [ORR & ABOWD 00].

exemple, consiste en un matériau déformable destiné à concevoir des maquettes de paysages, et sur lequel sont projetées des informations topographiques (figure 1.27, image de gauche). L'interaction se fait par modelage du matériau, dont la forme est capturée en temps réel par un scanner laser.

L'*informatique diffuse*[19] décrit une réalité augmentée où l'informatique, s'aidant de la miniaturisation et de la mise en réseau, se fond avec l'environnement quotidien jusqu'à être invisible et fournit des informations et des services à l'endroit et au moment opportuns [WEISER 91]. Ce nouveau paradigme d'*informatique embarquée* (voir section 1.2.3 page 11) met en avant une interaction naturelle entre l'utilisateur et l'environnement augmenté, mais surtout le remplacement des interactions explicites par une *sensibilité au contexte*, où des informations provenant de l'environnement (identité de l'utilisateur, localisation spatiale, objet d'intérêt, type d'activité) sont automatiquement collectées et interprétées. Ici, les dispositifs d'entrée sont remplacés par des capteurs. Le *sol intelligent* [ORR & ABOWD 00], par exemple, est un système non intrusif d'identification biométrique et de localisation basé sur des profils de charge (figure 1.27, image de droite).

1.4 L'interaction non-standard dans l'informatique grand public

Dans cette section, nous appliquons les enseignements tirés des sections précédentes à l'informatique grand public. Nous montrons que les dispositifs et les paradigmes d'interaction non standard ont un rôle non négligeable à jouer dans ce domaine, ce qui nous amènera naturellement à la notion d'interaction *adaptable*[20] au contexte.

Dans un premier temps, nous montrons l'intérêt d'une interaction adaptable, ouverte aux paradigmes non conventionnels, par opposition au modèle trop rigide de l'interaction standard. Nous définissons ensuite plus précisément la notion d'*adaptabilité en entrée*, dont nous nous servons par la suite pour établir un constat sur les applications interactives actuelles.

[19] Traduction de *Ubiquitous Computing*. Les traductions employées sont nombreuses : informatique *ubiquitaire*, *omniprésente*, *pervasive*, *disséminée*, ou encore *ubiquité numérique*.

[20] Par *adaptable*, nous n'entendrons pas « qui s'adapte automatiquement », mais plutôt « qu'il est possible d'adapter ».

1.4.1 Nécessité d'une interaction adaptable

Des contextes d'utilisation potentiellement nombreux

Nous avons mis en avant dans les sections précédentes l'intérêt d'employer des dispositifs d'entrée et des techniques d'interaction adaptés au contexte d'utilisation, en prenant notamment en compte les caractéristiques des utilisateurs potentiels (compétences ou limitations) et ceux de l'environnement (espace limité, bruyant, mobile, etc.).

Parmi les exemples que nous avons évoqué, les créations les plus originales étaient dédiées à des contextes d'utilisation très restreints : animation cinématographique ou planification neuro-chirurgicale, par exemple. Cela ne signifie pas que les techniques non conventionnelles n'aient d'intérêt que dans les applications professionnelles. Il est aussi avantageux dans les applications grand public d'adapter l'interaction au contexte, bien que les contextes d'utilisation *potentiels* soient extrêmement nombreux.

N'importe qui est susceptible d'utiliser un logiciel de gestion de courrier électronique, quel que soit son métier et ses compétences, et quelque soit son handicap. Il est probable que certaines personnes emploieraient des dispositifs et des techniques très personnelles, s'ils en avaient la possibilité. En outre, un tel logiciel est également susceptible d'être utilisé dans des environnements non conventionnels et sur des plate-formes autres que l'ordinateur de bureau : sur un ordinateur de poche, une borne interactive ou dans une voiture, par exemple.

Employer des techniques d'interaction efficaces

Chaque technique d'interaction est adaptée à un contexte d'utilisation particulier : l'interaction gestuelle, par exemple, est extrêmement efficace lorsqu'elle est associée à des dispositifs d'entrée de type stylet et reste exploitable avec des dispositifs de pointage moins précis comme la souris, mais son emploi est nettement moins pertinent avec un clavier. En outre, si l'interaction gestuelle est particulièrement intéressante dans les applications hautement graphiques, elle l'est moins dans les applications de type « calculatrice », du moins dans l'hypothèse où un clavier est présent. D'un autre côté, chaque dispositif physique « appelle » des tâches et des techniques d'interaction particulières : un clavier est particulièrement à l'aise dans les tâches de saisie textuelle, pour lesquelles il a été conçu, bien qu'il puisse également servir à d'autres fins.

Il existe ainsi des relations complexes du type « va bien avec » entre les types d'utilisateurs, les dispositifs d'entrée, les techniques d'interaction, les types de tâches et les types d'applications. De telles relations lient également entre elles les techniques d'interaction qui peuvent être combinées (par exemple, pointage + menu). Ces associations pourraient être explicitées dans une matrice à entrées multiples, mais le nombre de dimensions (supérieur à deux) et leur nature continue (rendant toute énumération ou classification excessivement simplificatrice) a de quoi décourager quiconque voudrait se lancer dans cette tâche avec le souci d'être exhaustif.

Malgré tout, dans chaque contexte où l'utilisateur, les dispositifs d'entrée présents et les tâches applicatives sont *fixés*, les techniques d'interaction constituent l'unique *degré de liberté* restant. L'adaptabilité en entrée, que nous définirons plus loin, consiste à exploiter au mieux ce degré de liberté en fournissant les meilleures techniques d'interaction possibles pour chaque situation.

Aujourd'hui encore, on ne saurait cependant dire quelle technique d'interaction est la mieux adaptée à un contexte donné. La recherche en interaction homme-machine propose régulièrement de nou-

velles techniques d'interaction toujours plus naturelles et efficaces, et ce la plupart du temps pour des dispositifs d'entrée et des tâches *déjà connus*. On saurait encore moins déterminer les techniques optimales pour les couples *(dispositif, tâche)* les moins conventionnels (par exemple, quelle est la meilleure façon d'éditer une cellule dans un tableur avec pour seuls dispositifs un potentiomètre linéaire et un microphone), dont la plus grande partie n'a d'ailleurs jamais été explorée. L'évolution constante de notre savoir sur l'interaction implique des systèmes interactifs *ouverts* plutôt que figés.

Une vraie accessibilité

L'accessibilité d'un système interactif repose en grande partie sur sa capacité à concilier les couples *(dispositif, tâche)* que nous venons d'évoquer. Le problème a été trop vite réglé par les nombreux outils d'*émulation*, qui se proposent d'imiter à partir d'un dispositif d'entrée non-conventionnel le comportement d'un dispositif d'entrée connu. Bien que parfois satisfaisante, cette solution ignore une dimension importante de l'interaction en entrée, à savoir que les propriétés intrinsèques d'un dispositif d'entrée influent grandement sur l'utilisabilité d'une technique d'interaction. Voici par exemple quelques conseils donnés par un fabricant d'écrans tactiles pour bornes interactives à leurs clients : *« Utilisez une simple interface à base de pointer-et-cliquer : pas de double-clic, pas de cliquer-glisser. Cachez le curseur, il est inutile si l'écran n'induit pas d'effet de parallaxe »* [ELO 02].

Proposer un vaste ensemble de techniques d'interaction prédéfinies pour permettre de sélectionner celle qui convient le mieux au contexte serait un pas en avant, mais ne résoudrait pas totalement le problème, car les techniques d'interaction constituent un *continuum* au même titre que les utilisateurs ou les dispositifs d'entrée. Il est vrai que les publications décrivant ça et là de nouvelles techniques d'interaction pourraient laisser croire qu'il existe à un instant donné un ensemble vaste mais dénombrable de techniques connues. Il y a cependant une différence importante entre une *description haut-niveau* d'une technique d'interaction telle qu'on en trouve dans la littérature scientifique ou dans les documentations et son *implémentation concrète*.

Pour illustrer ce fait, prenons l'exemple des menus hiérarchiques. Un manuel décrivant l'utilisation de cette technique pourrait contenir, entre autres, l'information suivante : « lorsque le sous-menu apparaît, vous pouvez pointer directement vers l'un de ses éléments ». Dans la plupart des menus hiérarchiques, un mécanisme de temporisation empêche en effet d'autres sous-menus de s'ouvrir pendant l'opération. Toutefois les délais ont été réglés pour une souris manipulée par un utilisateur moyen, et sont inadaptés à des utilisateurs plus lents ou des dispositifs d'entrée qui bornent la vitesse de pointage (voir figure 1.28 page suivante). Le fait que les menus hiérarchiques en général soient adaptés à tout type de dispositif de pointage ne signifie pas qu'une implémentation donnée sera utilisable avec n'importe lequel de ces dispositifs.

Une modification mineure ou une paramétrisation différente de la technique des menus hiérarchiques peut la rendre à nouveau utilisable dans le cas évoqué, bien que peut-être moins productive dans le cas courant. Plus généralement, il existe pour chaque technique d'interaction en apparence générique, un grand nombre d'implémentations concrètes (variantes, paramétrisations), chacune étant adaptée ou non à un contexte très spécifique. C'est pourquoi une réelle accessibilité implique des techniques d'interaction non seulement variées, mais dont on puisse également adapter le comportement de manière très fine pour pouvoir les optimiser en fonction de l'utilisateur, des dispositifs et des tâches.

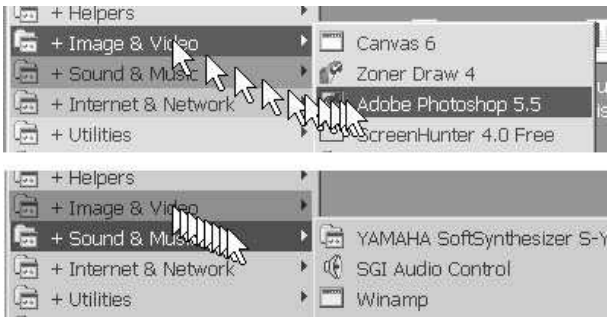


FIG. 1.28 – Un menu hiérarchique utilisé avec une souris (en haut) et avec un dispositif de pointage de type manette (en bas). Dans ce menu, un mécanisme de temporisation évite à la cible (ici, *Adobe Photoshop*) de disparaître pendant le pointage. Dans le second cas cependant, la disparition prématurée de la cible et l'ouverture d'un menu non souhaité ne peuvent être évités.

Personnalités et préférences individuelles

Il existe différents types d'utilisateurs. En premier lieu, ceux qui éprouvent une répulsion ou une peur de l'informatique s'opposent à ceux qui se sentent attirés par ce domaine [SHNEIDERMAN 98]. Même parmi ces derniers, des querelles amicales opposent les partisans de tel ou tel outil, système d'exploitation ou style d'interaction : ainsi, si certains puristes mettent en avant la puissance des commandes textuelles, les personnalités créatives penchent le plus souvent pour l'interaction graphique et les éditeurs WYSIWYG [21]. Des différences fondamentales existeraient également entre les hommes et les femmes, notamment en ce qui concerne les choix de jeux vidéo [SHNEIDERMAN 98].

Ces oppositions montrent, en partie seulement, à quel point les personnalités des utilisateurs de l'informatique sont diverses et les préférences individuelles variées. Une interface dédiée à l'utilisateur moyen ne pourra provoquer qu'une majorité d'insatisfaits. Chaque utilisateur étant unique, il est essentiel de prendre en compte ses préférences propres, notamment en lui permettant de personnaliser l'interaction pour l'adapter à ses préférences.

Certains utilisateurs sont prêts à consacrer du temps pour personnaliser et améliorer les outils informatiques qu'ils utilisent. Nous les qualifions d'*utilisateurs avancés*. Comparés aux utilisateurs moyens, les utilisateurs avancés ont souvent une meilleure connaissance de l'informatique. Les niveaux d'expertise sont cependant bien différents entre les amateurs passionnés, les professionnels qui utilisent les outils informatiques, et les informaticiens. C'est donc la *motivation* qui caractérise le mieux les utilisateurs avancés, qui sont prêts à fournir des efforts pour améliorer leur productivité.

La facilité de prise en main et la simplicité d'utilisation sont des critères essentiels dans les interfaces grand public, et il est d'usage notamment de ne pas encombrer l'utilisateur de détails et de choix inutiles. Cependant, parce que l'informatique connaît un développement important, les utilisateurs avancés constituent une catégorie croissante d'utilisateurs, et cette catégorie est malheureusement encore peu reconnue par les fabricants logiciels. Car comme nous le verrons par la suite, les interfaces actuelles n'offrent pas d'outils de configuration à la hauteur de ce que sont en droit d'attendre les utilisateurs avancés.

[21] What You See Is What You Get

Les faiblesses de l'interaction standard et leurs conséquences

La plupart des applications interactives grand public reposent essentiellement sur le *modèle de l'interaction standard* : une souris, un clavier, et un nombre restreint de techniques d'interaction associées (voir section 1.1.1 page 2 sur l'interaction standard). Ces dispositifs et ces techniques ont été conçus avec une idée de *généricité*. Cette généricité est profitable aux programmeurs car elle leur garantit une certaine *réutilisabilité* d'une application à l'autre, et également aux utilisateurs car elle garantit une certaine *cohérence* de l'interaction d'une application à l'autre. En outre, les techniques d'interaction génériques peuvent être conçues avec soin et testées une fois pour toutes. Cependant, ce modèle possède deux inconvénients de taille :

Les dispositifs et techniques d'interaction standard ne sont jamais totalement appropriées : ceux-ci visent un utilisateur moyen (utilisateur capable d'utiliser une souris et un clavier, et qui s'en contente), un environnement d'utilisation moyen (assis devant un bureau dans des conditions normales), et un ensemble de tâches moyennes (manipuler des widgets). Parce que ces techniques sont trop génériques, elles ne sont jamais totalement appropriées, ni au contexte d'utilisation, ni aux tâches de l'application.

Les dispositifs et techniques d'interaction standard peuvent être totalement inappropriées : c'est le cas en particulier lorsque la souris ou le clavier ne peuvent être employés, en raison de contraintes liées à l'utilisateur ou à l'environnement. Des plate-formes alternatives ont été proposées dans lesquelles l'interaction a été repensée avec d'autres dispositifs (interaction à base de stylet, par exemple). Cependant, les applications y sont toujours développées sur le même modèle, c'est-à-dire par-dessus un ensemble préétabli de techniques d'interaction, ce qui interdit toute portabilité d'une plate-forme à l'autre.

Enfin, le modèle de l'interaction standard, par sa rigidité, fige l'évolution de l'informatique grand public et l'empêche de suivre les innovations. De nouvelles techniques d'interaction sont publiées chaque année dans la littérature scientifique, et malgré leurs avantages, sont ignorées dans les applications commerciales. Seules certaines de ces applications se distinguent des logiciels conventionnels et sortent des sentiers tracés par le modèle de l'interaction standard. Bien que ces applications constituent encore une minorité et que leurs capacités soient encore très limitées, elles dénotent un courant positif dans l'informatique grand public, qui va dans le sens de notre discours.

Nous établirons pour compléter cette discussion un constat général sur les applications interactives actuelles, afin de mieux constater l'influence du modèle d'interaction standard sur l'informatique grand public.

1.4.2 Définition de l'adaptabilité en entrée

Pour parler de l'informatique grand public d'aujourd'hui, il est difficile de ne pas prendre pour base de comparaison les standards existants. Nous introduisons par conséquent les termes d'entrées *enrichies* ou *appauvries*, par référence aux entrées standard que constituent la souris et le clavier :

Les *entrées enrichies* décrivent les dispositifs d'entrée ou les combinaisons de dispositifs dont la bande passante est supérieure à celle des dispositifs standard. Les exemples vont des contrôles supplémentaires sur les dispositifs standard aux dispositifs multi-dimensionnels, à large bande passante, ainsi que les dispositifs multiples.

Les *entrée appauvries* décrivent les dispositifs d'entrée ou les combinaisons de dispositifs dont la bande passante est inférieure à celle des dispositifs standard. Les exemples vont d'un dispositif standard manquant à des entrées extrêmement pauvres comme le simple bouton.

Nous définissons l'*adaptabilité en entrée* comme une combinaison de trois propriétés : la contrôlabilité, l'accessibilité et la configurabilité.

- La **contrôlabilité** est la capacité d'un système interactif à exploiter efficacement des *entrées enrichies*, ou à utiliser les dispositifs standard de façon *plus efficace*. L'utilisation efficace d'entrées enrichies implique un usage approprié de la bande passante utile et des compatibilités tâche/dispositif. L'utilisation plus efficace des entrées standard implique l'emploi de techniques d'interaction plus concises et directes, de type Post-WIMP.
- L'**accessibilité** est la capacité d'un système interactif à exploiter des *entrées appauvries*. L'utilisation efficace d'entrées appauvries implique l'emploi de techniques d'interaction assez *riches* pour compenser la dégradation de la bande passante et de la compatibilité tâche/dispositif.
- La **configurabilité** exprime la capacité pour l'utilisateur de choisir librement *la manière* dont il veut utiliser ses dispositifs d'entrée pour contrôler l'application. Cela va de la sélection de dispositifs et de l'assignation d'actions à la spécification détaillée de techniques d'interaction riches. La facilité de configuration joue également un rôle crucial dans la configurabilité, tout comme le type d'utilisateurs visé et les expertises requises.

1.4.3 Adaptabilité des systèmes interactifs actuels

Dans cette section, nous décrivons l'adaptabilité en entrée des systèmes interactifs actuels. Nous distinguons deux grands niveaux d'adaptabilité : le niveau système et le niveau applicatif.

Au niveau système

Les dispositifs d'entrée sont gérés en premier lieu au niveau du système d'exploitation, qui offre des services de base aux applications interactives. L'interaction avec l'ensemble des applications interactives peut être personnalisée à ce niveau.

La plupart des systèmes d'exploitation comportent des panneaux de configuration qui permettent de personnaliser la gestion de la souris et du clavier (figure 1.29 page suivante). Sur les systèmes de type Microsoft Windows, l'utilisateur peut ainsi adapter l'interaction à son niveau d'expertise (vitesse du double-clic, délai de répétition des touches du clavier), ses particularités culturelles (choix de la langue du clavier) ou motrices (inversion des boutons de la souris pour gaucher) et, plus généralement, ses préférences individuelles (vitesse et apparence du pointeur, ouverture avec simple clic ou double clic, etc.). Enfin, des « options d'accessibilité » permettent de faciliter l'accès aux utilisateurs handicapés par des techniques simples comme le filtrage des erreurs de frappe ou la rémanence de touches facilitant l'usage de combinaisons.

Malgré les fonctionnalités offertes par les panneaux de configuration, les systèmes d'exploitation restent extrêmement rigides du point de vue des entrées. Voici une liste non exhaustive de leurs faiblesses :

Fusion des entrées Les dispositifs de pointage multiples et les claviers multiples sont fusionnés au niveau du système d'exploitation. Lorsque plusieurs dispositifs de pointage (souris, tablette gra-

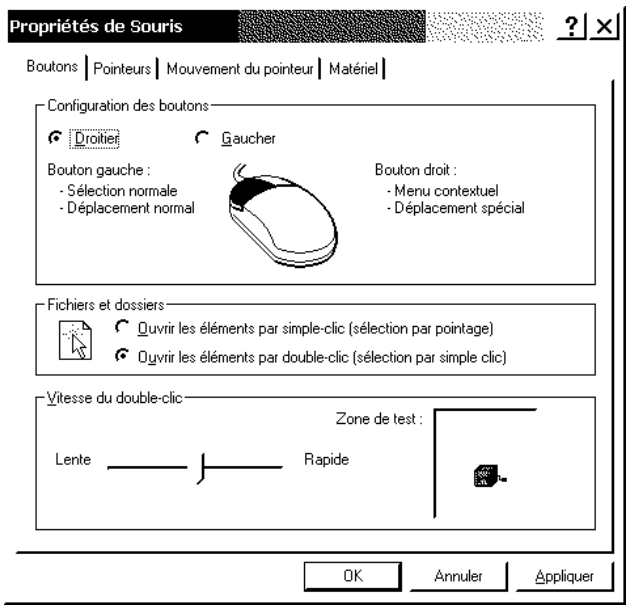


FIG. 1.29 – Configuration de la souris sous Microsoft Windows NT4.

phique) sont connectés, ils "poussent" tous le même curseur, ce qui rend l'interaction bimanuelle ou multi-utilisateurs impossible. Les claviers multiples sont vus comme un seul clavier. Windows, par exemple, ne permet pas d'affecter une langue à chaque clavier. La fusion des entrées a des conséquences négatives sur la *contrôlabilité*.

Entrées étendues ignorées : La plupart des dispositifs non-standard (manettes de jeux, dispositifs 3D) ne peuvent pas être utilisés pour interagir avec les applications classiques. Certains pilotes proposent un contrôle "par compatibilité", mais dans lequel les caractéristiques intéressantes du dispositif sont ignorées. Par exemple, une tablette graphique pourra servir à des tâches de pointage, mais sa résolution et sa sensibilité à la pression seront inutilisées. Le fait d'ignorer les entrées étendues a de fortes conséquences sur la *contrôlabilité*.

Entrées standard requises : La suppression de la souris ou du clavier rend l'interaction laborieuse, voire impossible. Les raccourcis-claviers ne sont pas conçus pour remplacer la souris, car leur utilisation exclusive est pénible, et parce qu'ils ne balayent pas l'ensemble des tâches possibles. Quant aux outils de saisie textuelle par pointage, ils sont inexistants ou peu efficaces (clavier-écran). L'interaction exclusivement gestuelle en particulier, n'est pas prise en compte dans les systèmes d'exploitation de type ordinateur de bureau. La nécessité d'employer les dispositifs standard a des conséquences néfastes sur l'*accessibilité*.

Configurabilité pauvre : Bien que parfois utiles, les panneaux de configuration des systèmes d'exploitation, mêmes surchargés, ne prennent en compte qu'un nombre fixe de cas déjà prévus à l'avance. Chaque pilote de dispositif propose des outils de configuration basés sur le même modèle, bien qu'adaptés au type de matériel qu'il prend en charge. La pauvreté de la *configurabilité* a naturellement des répercussions importantes sur l'*accessibilité*. Les options d'accessibilité dans les systèmes d'exploitation sont d'ailleurs anecdotiques car peu nombreuses, stéréotypées et souvent inadaptées. La plupart des techniques d'accessibilité courantes, basées sur l'émulation de la souris et du clavier,

ne sont accessibles qu'à travers un matériel ou des logiciels spécialisés.

Au niveau des applications

L'adaptabilité en entrée varie selon les applications. Nous distinguons quatre types d'applications courantes : les applications traditionnelles, les applications spécialisées, les jeux vidéo et les applications 3D.

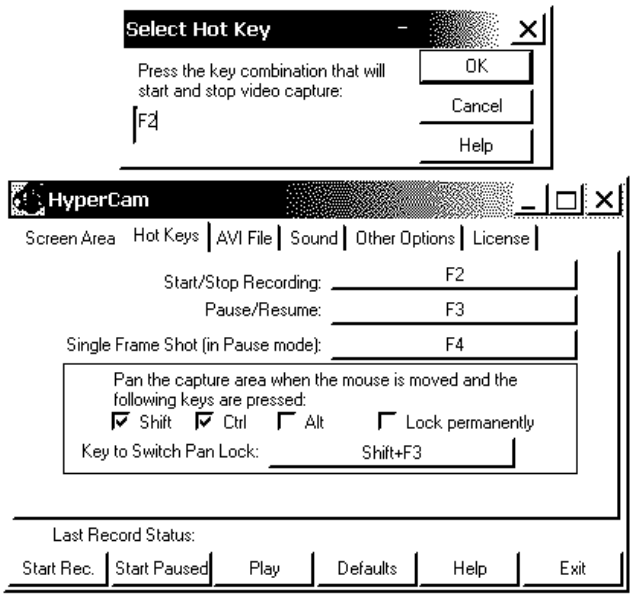


FIG. 1.30 – La configuration des touches dans un logiciel de capture d'écran.

Les applications traditionnelles : Les applications traditionnelles comme les logiciels de bureautique reposent exclusivement sur les services fournis par le système d'exploitation, ou sur des outils de développement qui reposent sur ces services. Leur adaptabilité en entrée est par conséquent limitée à celle précédemment décrite. Certaines de ces applications permettent à travers des options de menu de spécifier des préférences qui se limitent souvent, du point de vue des entrées, à la configuration des raccourcis clavier. Bien que marginales, des techniques d'interaction non standard commencent à apparaître dans certaines applications commerciales. C'est le cas des commandes gestuelles dans les navigateurs *Mozilla* [MOZILLA 03] et *Opera* [OPERA 03], qui offrent par conséquent une contrôlabilité améliorée par rapport aux applications conventionnelles.

Les applications spécialisées : Certaines applications semi-professionnelles comme les logiciels d'infographie ou de composition musicale gagnent particulièrement à être utilisées avec des dispositifs spécialisés. Ces dernières peuvent être en partie contrôlés par des instruments de musique MIDI. Le logiciel de dessin *Adobe Photoshop* gère les tablettes graphiques de façon spécifique, et exploite leur résolution élevée ainsi que leur sensibilité à la pression [WACOM 03]. Cependant, la configurabilité est limitée (la pression ne peut être assignée qu'à la taille de la brosse, son opacité ou sa couleur) et les autres dimensions comme l'inclinaison (*tilt*) sont ignorées. Ces applications spécialisées offrent une meilleure contrôlabilité grâce à la prise en charge d'un type de dispositif spécifique en plus des dispositifs standard, mais leur utilisation reste très classique et ils ne sont ni excessivement contrôlables, ni

excessivement configurables, et encore moins accessibles.

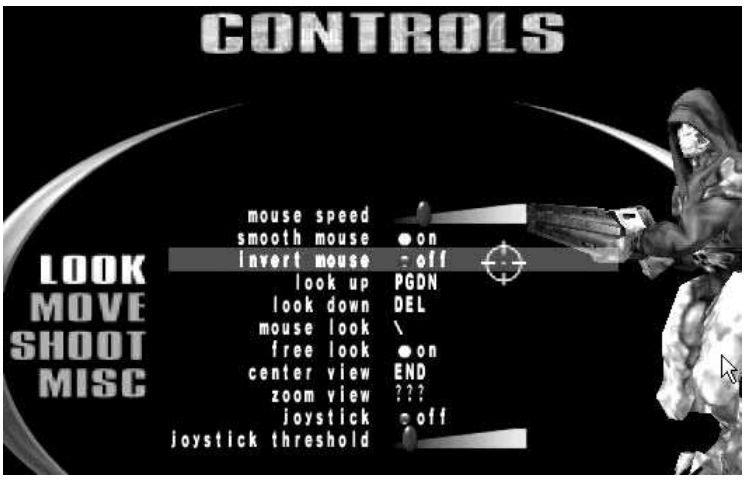


FIG. 1.31 – Configuration des entrées dans le jeu *Quake III* [HONEYWELL 99].

Les jeux vidéo : Les jeux vidéo supportent des dispositifs d'entrée de plus en plus sophistiqués, tels que les manettes à boutons multiples et les dispositifs de simulation. Traditionnellement, ils offrent à l'utilisateur le choix entre le clavier, la manette et la souris pour contrôler le jeu. La possibilité de personnaliser les contrôles est un aspect important des jeux vidéo (figure 1.31). Ainsi les boutons ou les touches sont librement assignables. Cependant l'utilisateur a moins de contrôle sur la façon dont les dimensions continues sont employées, et peut tout au plus jouer sur la sensibilité ou l'inversion des axes. En outre, les dispositifs non-standard qui ne sont pas dédiés aux jeux ne sont pas pris en compte. Les jeux vidéo sont bien plus contrôlables que les applications traditionnelles, mais ils sont à peine plus configurables, et l'accessibilité n'est pas une préoccupation majeure dans ce secteur.

Les applications 3D : Les logiciels d'animation et de modélisation 3D comme *WorldUp* [SENSE8 03] ou *Vega Prime* [PARADIGM 03] supportent un grand nombre de dispositifs sophistiqués, tels que les souris 3D ou les capteurs à six degrés de liberté. En général, chaque dimension peut être librement assignée à des attributs d'objets 3D, ce qui offre des interactions à la fois riches et configurables. Certaines applications comme *Virtools* [VIRTOOLS 01] possèdent un éditeur graphique d'interaction qui, bien que complexe d'utilisation, multiplie les possibilités de configuration (voir la section 2.7 page 79 pour une description de ces outils). Mais là encore, seuls les dispositifs et les techniques d'interaction 3D sont pris en compte. Les applications 3D sont néanmoins les plus contrôlables et les plus configurables, bien que l'accessibilité ne soit pas non plus une préoccupation majeure dans ce domaine.

1.5 Conclusion

Nous avons montré dans ce chapitre l'intérêt de l'interaction en entrée non standard, d'abord en décrivant des situations concrètes où l'emploi de dispositifs d'entrée alternatifs dédiés à l'utilisateur, à la tâche ou à l'environnement est inévitable ou permet d'améliorer considérablement la qualité de l'interaction. Puis, en présentant les principaux nouveaux paradigmes d'interaction, que l'on regroupe sous le nom d'interfaces Post-WIMP, et qui autorisent des interactions plus concises, directes, parallèles et naturelles.

Ces techniques sont peu connues à ce jour, et la plupart n'ont été pas été exploitées ailleurs que dans des prototypes de recherche, ou ont été conçues pour des situations professionnelles très particulières. Étant donné la démocratisation des dispositifs non standard, beaucoup d'entre elles pourraient cependant être employées dans l'informatique grand public, dans laquelle les utilisateurs, les environnements de travail, et les tâches potentielles sont extrêmement variées.

À partir de ce constat, nous avons défini pour les applications et les systèmes interactifs grand public une propriété qui nous semble essentielle, celle de l'*adaptabilité en entrée*. L'adaptabilité en entrée est la faculté de prendre en compte à la fois des entrées *enrichies* et *appauvries*, d'employer des *techniques d'interaction adaptées* à chacune de ces situations, et d'offrir à l'utilisateur le choix de ces techniques ainsi que la possibilité de les *configurer* finement.

Nous avons par la suite évalué l'adaptabilité en entrée des applications interactives actuelles, et constaté une démarcation importante entre les applications interactives conventionnelles et les applications comme les jeux vidéo et les applications 3D, pour lesquelles la compatibilité avec les dispositifs non-standard est un critère commercial important. Dans l'ensemble cependant, les applications que nous connaissons sont toutes extrêmement limitées dans leur adaptabilité en entrée. C'est le cas aussi bien des applications grand public que des applications à usage semi-professionnel, et cela reste vrai pour les applications plus spécialisées.

La raison principale est que ces applications reposent sur des services et des outils qui s'appuient à leur tour sur le modèle rigide de l'interaction standard, et qui font de l'adaptabilité en entrée un objectif extrêmement difficile à atteindre. L'implémentation de nouvelles techniques d'interaction et l'exploitation des dispositifs d'entrée non-standard, en particulier, sont des tâches extrêmement ardues pour le programmeur. De nombreux modèles et outils pour l'Interaction-Homme-Machine ont été néanmoins étudiés et proposées par la recherche, et le sont encore aujourd'hui. Ces modèles et outils feront l'objet du prochain chapitre.

Chapitre 2

Modèles et outils pour l'interaction en entrée

Sommaire

2.1 Introduction

Une interface graphique est un système complexe, difficile à appréhender dans son ensemble, et qui nécessite d'être décomposé. Il existe de multiples stratégies de décomposition, dont certaines ont été acceptées comme références, et que nous présentons dans un premier temps. Contrairement aux modèles de référence, les modèles de type formel décrivent les interfaces ou des aspects particuliers de ces interfaces de manière particulièrement détaillée : nous présentons ceux qui nous semblent pertinents du point de vue de l'interaction en entrée. Nous décrivons ensuite deux types de modèles qui traitent deux aspects essentiels de l'interaction en entrée : les modèles de dispositifs, dont l'objectif est d'extraire les caractéristiques pertinentes des différents types de dispositifs d'entrée, et les modèles d'interaction, qui décrivent des règles et des principes généraux pour la conception d'interfaces naturelles et efficaces.

Si les modèles permettent de mieux comprendre l'interaction et sont d'une certaine aide dans la phase de conception, les outils de développement constituent les supports concrets pour l'implémentation des applications interactives. Certains de ces outils s'inspirent de modèles existants et les autres, largement majoritaires, emploient un modèle d'interface spécifique. Le développement d'applications interactives repose en grande partie sur les boîtes à outils graphiques traditionnelles, optimisées et câblées pour l'interaction en entrée standard. D'autres boîtes à outils graphiques, moins connues, prennent en charge des dispositifs d'entrée ou des techniques d'interaction non conventionnelles. Nous présentons ces dernières de façon détaillée, et discutons de leurs apports respectifs par rapports aux boîtes à outils traditionnelles. Pour finir, nous présentons les différents types d'éditeurs d'interaction, outils généralement destinés à un plus vaste public que les programmeurs, et qui permettent de spécifier tout ou partie de l'interaction en entrée.

Dans cet état de l'art, nous avons choisi de détailler et d'expliquer les travaux représentatifs de chacune de ces approches plutôt que de produire des longues listes de références. Les points-clés de ce chapitre seront en outre résumés en guise de conclusion.

2.2 Les modèles d'interface de référence

La recherche en interaction homme-machine a consacré beaucoup d'énergie à développer des modèles génériques et abstraits de systèmes interactifs. L'objectif de ces *modèles de référence* est d'obtenir une meilleure compréhension des systèmes interactifs existants, mettre en place une base commune de communication sur le domaine, et guider le choix d'une architecture logicielle lors du développement de nouveaux systèmes interactifs.

Chaque modèle se propose d'identifier les éléments significatifs qui entrent dans la composition de la plupart des systèmes interactifs, ainsi que les relations qui les lient. Bien que les terminologies employées diffèrent sensiblement, on retrouve souvent d'un modèle à l'autre les mêmes principes de base.

Tous les modèles partent du principe qu'un système interactif comporte une partie "interface" et une partie "application pure" (figure 2.1 page ci-contre). Cette dernière est souvent appelée *noyau fonctionnel*, et tout ce qui s'y réfère appartient au *domaine*. Le noyau fonctionnel est considéré comme préexistant, et les modèles de systèmes interactifs décrivent essentiellement la partie interface, ainsi que ses relations avec les objets du domaine.

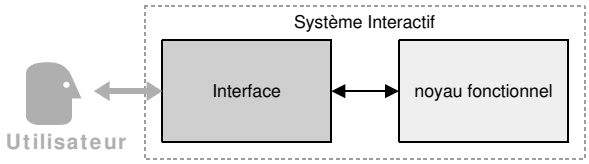


FIG. 2.1 – Modèle primitif d'un système interactif.

La plupart des modèles identifient au moins trois types d'éléments dans la composition des interfaces, et distinguent un "coté utilisateur" et un "côté noyau fonctionnel". Il comprennent presque toujours des éléments en contact direct avec l'utilisateur (*présentations*), des éléments en contact direct avec le noyau fonctionnel ou qui en font partie (*interfaces du noyau fonctionnel, abstractions, modèles*), et des éléments articulatoires (*contrôleurs, adaptateurs*).

Malgré ces points communs, certains modèles procèdent d'approches très différentes, et on distingue communément deux grands groupes de modèles de référence. Les modèles *linguistiques* ou modèles *à couches* décrivent la structure globale d'une application interactive sous forme de couches logiques. Ces modèles sont également appelés modèles *logiques*, ou sont qualifiés d'*abstraits*. Les seconds types de modèles sont les modèles *à agents* ou *à interacteurs*, ou encore modèles *orientés objet*. Ces modèles s'inspirent du paradigme de programmation par objets, et proposent une décomposition modulaire de l'interface en un ensemble d'agents communicants.

Nous décrivons dans la suite ces deux groupes de modèles, ainsi que leurs principaux représentants.

2.2.1 Les modèles linguistiques

Les modèles de référence linguistiques se basent sur une *approche linguistique* de l'interaction, inspirée des architectures de compilateurs. L'approche linguistique identifie trois aspects dans l'interaction :

- *Aspects lexicaux* : ces aspects désignent tout ce qui peut être assimilé à un vocabulaire d'entrée (clic, glisser), ou de sortie (ensembles d'icônes).
- *Aspects syntaxiques* : ils peuvent désigner des grammaires d'entrée représentant les séquences d'actions valides, ou les aspects spatiaux et temporels de l'affichage.
- *Aspects sémantiques* : ils correspondent à la partie fonctionnelle de l'application, qui détermine en dernier lieu le sens d'une action et génère les erreurs.

À défaut de donner des indications précises sur la façon dont doit être structuré et implémenté un système interactif, les modèles linguistiques identifient les couches logiques qui doivent y apparaître.

Dans cette section, nous décrivons dans leurs grandes lignes et de façon chronologique les principaux modèles linguistiques, à savoir le modèle de Seeheim, le modèle Arch, ainsi que son métamodèle Slinky.

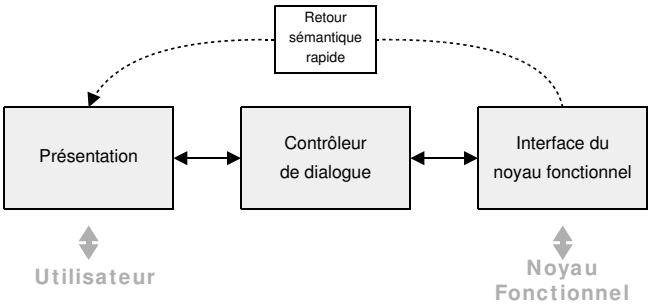


FIG. 2.2 – Le modèle de Seeheim.

Seeheim

Le premier modèle de référence largement accepté part d'une approche linguistique. Celui-ci est issu d'un groupe de travail sur les systèmes interactifs ayant eu lieu à Seeheim en 1985 [PFAFF 85]. Le modèle de Seeheim était principalement destiné au traitement lexical des entrées et sorties dans les interfaces textuelles. S'il est peu utile aujourd'hui pour décrire les systèmes interactifs hautement graphiques, il a servi de base à beaucoup d'autres modèles.

Le modèle de Seeheim[PFAFF 85] propose de diviser l'interface en trois couches logicielles selon une approche linguistique (figure 2.2) :

- *La présentation* est la couche en contact direct avec les entrées et sorties. Elle interprète les actions de l'utilisateur (dispositifs physiques) et génère les sorties (affichage) au niveau lexical.
- *Le contrôleur de dialogue* gère le séquencement de l'interaction en entrée et en sortie. Il maintient un état lui permettant de gérer les modes d'interaction et les enchaînements d'écrans.
- *L'interface du noyau fonctionnel* est la couche adaptative entre le système interactif et le noyau fonctionnel. Elle convertit les entrées en appels du noyau fonctionnel et les données abstraites de l'application en des objets présentables à l'utilisateur.

Un autre élément a été ajouté au modèle de Seeheim pour prendre en compte le *retour sémantique rapide* : par exemple, lors d'une opération de glisser-déposer, il peut s'avérer utile de modifier instantanément l'apparence de l'icône-cible pour indiquer si l'opération est valide. Dans ce cas, la couche de présentation doit court-circuiter le contrôleur de dialogue pour accéder directement aux informations sémantiques du noyau fonctionnel.

Arch / Slinky

Le modèle Arch [UIMS 92] affine le modèle de Seeheim en s'inspirant davantage des boîtes à outils graphiques actuelles. Ce modèle incorpore la notion de plate-forme de développement, et décrit la nature des données qui transitent entre les différents composants.

Le modèle Arch identifie cinq composants dans les systèmes interactifs (figure 2.3 page ci-contre) :

- *Le composant d'interaction* désigne l'ensemble des widgets (objets interactifs) d'une boîte à outils, ainsi que les communications avec les périphériques physiques. Il dépend d'une plate-forme donnée.

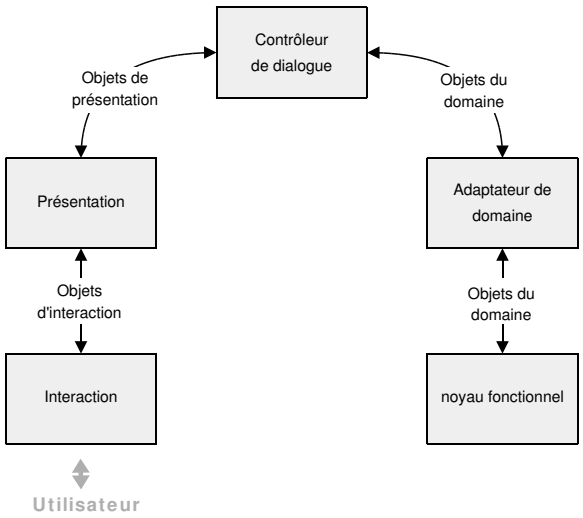


FIG. 2.3 – Le modèle Arch.

- *Le composant de présentation* joue le rôle de médiateur entre le composant d'interaction et le contrôleur de dialogue. Il maintient une représentation logique des widgets qui est indépendante de la plate-forme.
- *Le contrôleur de dialogue* joue le même rôle articulatoire que celui du modèle de Seeheim. Il est notamment responsable du séquencement des tâches et du maintien de la consistance entre les vues multiples.
- *L'adaptateur du domaine* est un composant médiateur entre le contrôleur de dialogue et le noyau fonctionnel. Il est responsable des tâches dépendantes du domaine qui ne font pas partie du noyau fonctionnel mais qui sont nécessaires à sa manipulation par l'utilisateur. Ces tâches comprennent la réorganisation des données du domaine dans des buts de manipulation interactive, ou la détection des erreurs sémantiques.
- *Le noyau fonctionnel représente la partie non interactive de l'application.*

Le *composant d'interaction* et le *noyau fonctionnel* constituent les deux pieds de l'arche. Les autres composants décrivent la partie de l'interface qui doit être implémentée pour faire le lien entre la partie purement fonctionnelle d'une application et les widgets et les événements de la boîte à outils.

Le modèle Arch introduit également trois types d'objets décrivant la nature des informations qui transitent entre les composants (figure 2.3) :

- *Les objets du domaine* contiennent des données provenant directement ou indirectement du noyau fonctionnel (par exemple, le résultat d'une requête dans une base de donnés).
- *Les objets de présentation* sont des objets d'interaction virtuels qui décrivent les événements produits par l'utilisateur et les données qui lui sont présentées, indépendamment des méthodes d'interaction et de visualisation.
- *Les objets d'interaction* sont des instances propres à une boîte à outils, et qui implémentent des techniques d'interaction et de visualisation spécifiques.

Dans la réalité, le poids relatif des composants Arch et la répartition de leurs fonctionnalités peut varier selon les types d'applications interactives et les choix et impératifs fixés par leurs déve-

FIG. 2.4 – Le jouet Slinky ayant inspiré le modèle du même nom.

loppeurs. Un métamodèle baptisé Slinky [UIMS 92] a été conçu au-dessus du modèle Arch afin de suggérer l'existence de ces variantes du modèle Arch. Le métamodèle Slinky, inspiré du jouet de même nom (figure 2.4), met notamment en évidence les notions de choix et de compromis inhérents au développement des applications interactives.

2.2.2 Les modèles à agents

En décomposant les interfaces en objets de même nature, les modèles à agents sont proches des langages à programmation objets et des interfaces à manipulation directe modernes. Dans cette section, nous décrivons les princpaux modèles à agents, à savoir MVC, PAC et le modèle hybride PAC/Amodeus.

MVC

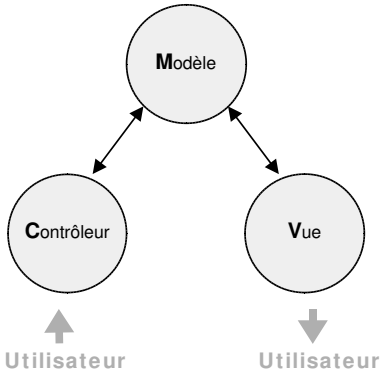


FIG. 2.5 – Le modèle MVC.

Le modèle MVC (Modèle, Vue, Contrôleur) [SCHMUCKER 86, KRASNER & POPE 88] a été introduit comme architecture de référence dans l'implémentation des interfaces utilisateur de l'environnement Smalltalk [GOLDBERG & ROBSON 81]. L'approche de MVC inspirera toute une lignée de modèles à base d'agents, dont les principales motivations sont la modifiabilité et la conception itérative, ainsi que la compatibilité avec les langages à objets.

MVC décompose les systèmes interactifs en une hiérarchie d'agents. Un agent MVC consiste en un *modèle* muni d'une ou plusieurs *vues*, et d'un ou plusieurs *contrôleurs* :

- *Le modèle* est le noyau fonctionnel de l'agent. Il peut représenter des données brutes (entier, chaîne de caractères) ou des objets ayant un comportement complexe. Le modèle notifie les vues qui lui sont associées à chaque fois que son état se trouve modifié par le noyau de l'application ou par ses contrôleurs.
- *La vue* maintient une représentation du modèle perceptible par l'utilisateur, qu'elle met à jour à chaque changement d'état du modèle. Elle est en général constituée d'objets graphiques. La figure 5 donne plusieurs exemples de vue pour un modèle simple.
- *Le contrôleur* reçoit et interprète les événements utilisateur, en les répercutant sur le modèle (modification de son état) ou sur la vue (retour instantané).

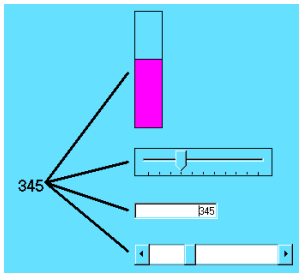


FIG. 2.6 – Plusieurs exemples de *look & feel*s pour un modèle consistant en une valeur entière et son domaine.

Chacun des trois composants de la triade MVC est un objet à part entière. Au sens de la programmation orientée objet, une classe de *Modèle* peut être compatible avec plusieurs classes de *Vues* et de *Contrôleurs*. Les composants interactifs d'une boîte à outils (widgets) sont la plupart du temps livrés avec un ensemble de paires *Contrôleur/Vue*, appelées *look & feel*s. Un entier comportant une valeur minimale et une valeur maximale peut être ainsi représenté par une jauge, un curseur, un champ de saisie, ou une barre de défilement (figure 2.6).

PAC

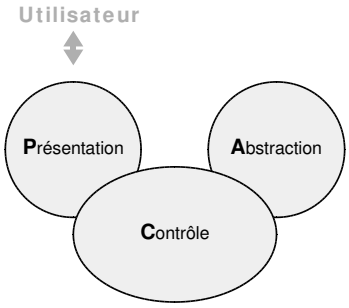


FIG. 2.7 – Le modèle PAC.

Par rapport à MVC, le modèle à agents PAC (Présentation, Abstraction, Contrôle) [COUTAZ 87] ré-introduit la vue et le contrôleur dans un objet monolithique mais rend explicite la synchronisation

du modèle et de la vue/contrôleur. Il propose en outre une méthode de description récursive qui étend le paradigme à agents avec la notion de couche d'abstraction.

Comme MVC, le modèle PAC décrit les systèmes interactifs comme une hiérarchie d'agents composés de trois modules. Mais ici, le rôle de ces modules diffère (figure 2.7 page précédente) :

- *La présentation* définit le comportement en entrée et en sortie de l'agent, tel qu'il est perçu par l'utilisateur.
- *L'abstraction* encapsule la partie sémantique de l'agent.
- *Le contrôle* maintient la consistance entre la présentation et l'abstraction, et communique avec les autres agents.

Notons qu'ici le composant d'abstraction est l'équivalent du composant modèle de MVC, et que la présentation correspond à une fusion des composants vue et contrôleur. Le composant contrôle n'a pas d'existence explicite dans le modèle MVC.

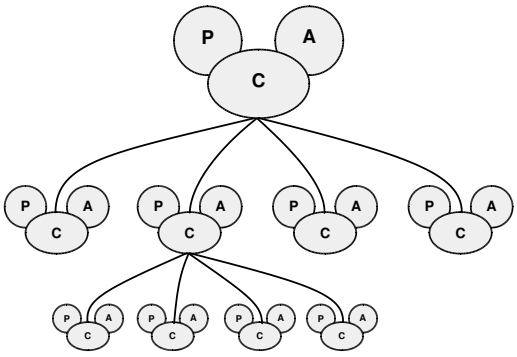


FIG. 2.8 – Hiérarchie d'agents PAC.

Par une approche récursive, le modèle PAC peut être appliqué de manière consistante à plusieurs niveaux d'un système interactif. Une application interactive peut ainsi être décrite comme une hiérarchie d'agents disposés sur plusieurs couches d'abstractions (figure 2.8). Ce type de représentation unifie en quelque sorte les modèles à agents et les approches à couches de type Seeheim.

PAC/Amodeus

PAC/Amodeus [NIGAY & COUTAZ 93] finit de réconcilier les approches linguistiques et à agents en proposant un modèle hybride. Il réutilise le modèle Arch et décrit son contrôleur de dialogue avec une hiérarchie d'agents PAC (figure 2.9 page ci-contre). L'adaptateur de domaine et la présentation de l'Arch communiquent directement avec chaque agent PAC à travers son abstraction (pour le premier) et sa présentation (pour le second).

L'objectif de PAC/Amodeus est de combiner les avantages du modèle Arch, qui intègre des aspects de génie logiciel comme la modifiabilité et la portabilité, et du modèle PAC, qui permet de structurer efficacement le contrôleur de dialogue jusque-là monolithique. Le modèle PAC/Amodeus a été employé pour modéliser des plate-formes multimodales [NIGAY & COUTAZ 95]. Les entrées sont décrites par des symboles et des grammaires, et des mécanismes de fusion de haut-niveau sont implémentés dans le dialogue par des agents PAC.

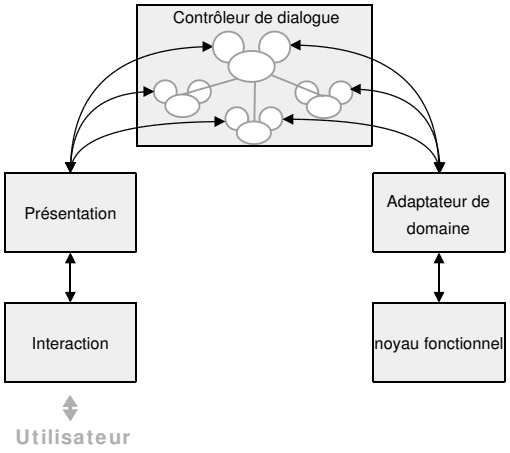


FIG. 2.9 – Le modèle PAC/Amodeus

2.2.3 Conclusion

Nous avons présenté dans cette section les principaux modèles d'interfaces de référence, dont l'objectif est de fournir des stratégies de décomposition fonctionnelle et structurelle pour les interfaces utilisateur afin de simplifier leur conception et constituer un support de raisonnement.

Les modèles linguistiques décomposent les interfaces en un petit nombre de couches, chaque couche possédant un rôle spécifique et traduisant un niveau d'abstraction. Cette approche est encore utile pour comprendre le fonctionnement de l'interaction, mais elle trop monolithique, abstraite et peu structurante. En outre, cette approche qui trouve son origine dans les interfaces textuelles est de plus en plus difficile à appliquer telle quelle aux interfaces modernes et encore plus aux interfaces Post-WIMP. C'est en particulier le cas de Seeheim. L'approche Arch est néanmoins plus pragmatique, car elle s'inspire des architectures concrètes des systèmes interactives, avec les notions de plate-forme et de boîte à outils.

Les modèles à agents décrivent un autre style de décomposition pour les interfaces utilisateur (dite aussi *horizontale* par opposition aux modèles à couches) : ces agents modélisent le système de façon homogène. Cette approche est particulièrement adaptée aux styles de programmation par objets, et permet de décrire des aspects tels que la modularité, le parallélisme et la distribution. La nécessité, cependant, de prendre en compte l'hétérogénéité des systèmes interactifs a conduit au modèle PAC/Amodeus. PAC/Amodeus emprunte à Arch des principes de génie logiciel essentiels telles que la modifiabilité et la portabilité, incarnées par ses adaptateurs de domaine et son composant de présentation.

Malgré leurs apports, les modèles de référence ne sont pas d'une grande aide pour décrire les aspects bas-niveau de l'interaction, aussi bien du point de vue sorties qu'entrées, ces aspects étant systématiquement encapsulés dans des blocs monolithiques. Rien n'est dit non plus sur les techniques d'interaction. Arch et PAC/Amodeus, en particulier, ne décrivent pas la façon de répartir les techniques d'interaction entre le composant de Présentation, le composant d'Interaction, et leur protocole de communication.

MVC est le seul modèle à dissocier la gestion des entrées de celle des sorties dans chacun de ses

agents, ce qui permet conceptuellement de décrire l'apparence ou le comportement en entrée d'un objet interactif de façon indépendante. Si cette séparation est très avantageuse du point de vue de la modifiabilité, elle est souvent difficile à mettre en œuvre : dans la plupart des composants visuels, la façon dont les entrées sont interprétées est très liée à l'affichage. MVC ne décrivant pas de protocole de communication entre les deux composants, la plupart des boîtes à outils existantes préfèrent fusionner la vue et le contrôleur en une entité unique [FOWLER 99].

2.3 Les modèles d'interface formels

Une grande variété de modèles, langages et méthodes formelles ont été développés pour spécifier, concevoir, vérifier, implémenter et tester divers systèmes. Certains d'entre eux ont été employés dans le but de concevoir des interfaces plus *fiables*, c'est-à-dire moins sujettes aux erreurs de conception humaines. Ces méthodes sont principalement employées dans les applications critiques, par exemple celles qui mettent en jeu la vie ou la sécurité humaine (centrales nucléaires, contrôle aérien). Elle sont également employées dans certaines applications industrielles non critiques car elles donnent des garanties de fiabilité tôt dans le cycle de conception et permettent de raccourcir la phase de test. Les modèles formels sont également utilisés comme support pour raisonner sur les propriétés et le comportement des interfaces.

Les méthodes formelles ne constituent pas le sujet de notre thèse. Cependant, ce sont également des *modèles* d'interfaces, et nous les décrivons comme tels. Nous présentons ici deux approches qui nous semblent essentielles dans le domaine de l'interaction homme-machine : les modèles à base de *systèmes de transition* et les modèles d'*interacteurs formels*.

2.3.1 Les systèmes de transition

Les *systèmes de transition* désignent un ensemble des notations permettant de décrire des comportements dits *« orientés contrôle »*. Il en existe un très grand nombre, presque tous des extensions des *automates à états finis* ou des *réseaux de Petri* respectivement introduits en 1955 et 1962.

Ces deux notations graphiques et leurs variantes sont encore couramment employées aujourd'hui en interaction homme-machine, soit à des fins de spécification formelle, soit pour raisonner sur des systèmes interactifs ou expliquer leur fonctionnement. S'ils ne permettent de décrire que des petites parties d'une interface (et ne constituent pas des *modèles d'interfaces* à proprement parler), ils entrent dans la composition de certaines méthodes formelles plus exhaustives. Nous présentons ici ces deux notations et quelques-uns de leurs emplois.

Les automates à états finis

Les *automates à états finis* ou machines à états finis, dont les principales variantes sont les machines de Moore [MOORE 56] et de Mealy [MEALY 55], ont été employés pour spécifier le comportement dynamique de certains systèmes, et sont notamment utilisés pour décrire certaines parties des interfaces utilisateur.

Les automates à états finis sont représentés graphiquement par des *diagrammes de transition d'états*, graphes orientés dont les nœuds sont des *états* et les arcs des *transitions* (figure 2.10). Un

état est un ensemble de valeurs qui caractérise le système à un moment donné dans le temps. Une transition d'état est une relation entre deux états indiquant un changement d'état possible, et qui peut (dans les *automates étendus*, par exemple) être annotée pour indiquer les conditions et les sources de déclenchement (*événements*) et les opérations qui en résultent (*sorties*).

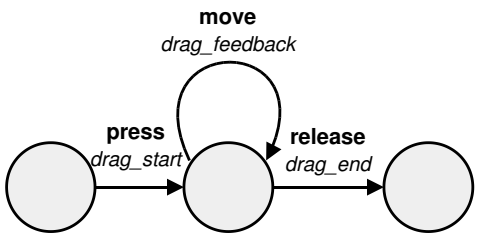


FIG. 2.10 – Le cliquer-glisser décrit par un diagramme de transition d'états [TYSON R. *et al.* 90].

La figure 2.10 est un diagramme de transition d'états simple et classique qui décrit la gestion du cliquer-glisser (*drag_start*, *drag_feedback*, *drag_end*) à partir d'événements souris de type *press*, *release* et *move* [TYSON R. *et al.* 90]. Les transitions sont annotées par les événements qui les déclenchent (en gras) et les sorties produites (en italique). Cet automate filtre tous les événements *move* ayant lieu avant le *press* ou après le *release*.

Les automates à états finis menant rapidement à une explosion du nombre d'états et de transitions, de nombreuses extensions ont été proposées. Les *machines à états hiérarchiques* et les *statecharts* [HAREL 87] décrivent des automates composables, ce qui facilite la description de systèmes complexes et autorise la réutilisabilité. Les automates à états finis peuvent également être employés comme compléments à des outils de programmation conventionnels [BLANCH 02] ou en association avec d'autres formalismes pour décrire des parties plus importantes de l'interface. Robert Jacob[JACOB *et al.* 99] emploie des automates pour spécifier les aspects discrets de l'interaction en entrée et des flots de données pour les aspects continus. Il décrit également un éditeur visuel VRED qui sera évoqué dans la section 2.7.3 page 88.

Les réseaux de Petri

Les *réseaux de Petri* [PETRI 62] sont une généralisation des automates à états. Ils ont été largement utilisés pour modéliser la concurrence et la synchronisation dans les systèmes distribués, et sont employés pour décrire certains comportements dans les interfaces. Nous présentons cette notation sans rentrer dans les détails.

Un réseau de Petri est un graphe biparti alterné qui possède deux types de nœuds : les *places* (cercles) et les *transitions* (rectangles). Des *arcs* (flèches) relient les places aux états (voir figure 2.11 page suivante). L'état du système, nommé *marquage*, est défini par la répartition de *jetons* (petits cercles foncés) dans les places. Une transition est *franchissable* sous certaines conditions, notamment lorsque suffisamment de jetons sont présents dans ses places d'entrée. Le franchissement d'une transition se traduit par une modification du marquage consistant la plupart du temps en un « déplacement » de jetons.

Le modèle des *transducteurs formels* [ACCOT *et al.* 97] emploie des *réseaux de Petri de haut-niveau* [JENSEN 95] pour décrire les transformations successives des événements d'entrée. La figure 2.11 page suivante

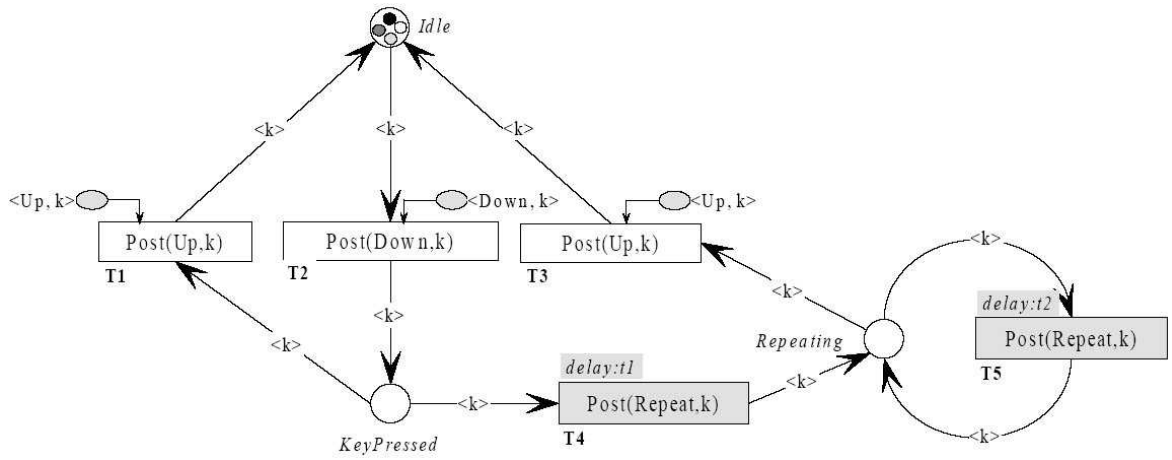


FIG. 2.11 – Un comportement bas-niveau du clavier (répétition de touches) décrit avec une variante élaborée de réseau de Petri [ACCOT *et al.* 97].

décrit un niveau de transformation pour le clavier. Dans cette notation, les jetons sont typés et comportent une valeur, les arcs sont annotés par des variables (<k> sur la figure) et des événements extérieurs viennent déclencher des transitions (ellipses grises et flèches coudées). Quatre touches sont représentées par quatre jetons en haut de la figure, en attente dans la place *Idle*. En-dessous se trouve la transition *Post*, qui est activée par des événements clavier de type *Down* avec une touche *k* en paramètre. Lorsqu'un événement $< Down, k >$ arrive, cette transition est franchie si le jeton *k* est présent dans $Idle$, auquel cas le même événement $< Down, k >$ est généré et le jeton passe dans la place *KeyPressed*. Il retourne à la place $Idle$ lors du prochain événement Up. Une partie de ce réseau de Petri répète simplement les événements *Down* et Up (avec cependant un filtrage grammatical), et une autre partie (la partie grisée) décrit un mécanisme de répétition automatique de touches avec génération d'événéments $Repeat$. Cette dernière emploie des *réseaux de Petri stochastiques généralisés* [MARSAN *et al.* 95] pour décrire des transitions temporisées.

Les réseaux de Petri ont été intégrés dans plusieurs méthodes et outils formels utilisés pour décrire la majeure partie des interfaces utilisateur. Le formalisme *ICO* (Interactive Cooperative Objects) [PALANQUE & BASTIDE 93, PALANQUE & BASTIDE 97] emploie une approche orientée-objet pour modéliser les aspects statiques du système interactif et des réseaux de Petri de haut-niveau pour en décrire les aspects dynamiques. Il est livré avec l'outil *PetShop* [BASTIDE *et al.* 02, SCHYN *et al.* 03], qui comporte un éditeur interactif permettant de construire des spécifications *ICO* et de les exécuter.

2.3.2 Les interacteurs formels

Dans cette section, nous regroupons sous le terme *interacteurs formels* deux modèles formels très liés, et qui décomposent les systèmes interactifs en unités autonomes appelées *interacteurs* (objets d'interaction), qui communiquent entre eux, avec le système ou avec l'utilisateur par le biais de stimuli (ou événements).

Les deux principaux modèles formels à base d'interacteurs ont été développés dans le cadre du projet Esprit AMODEUS, l'un à l'Université d'York (Angleterre), l'autre à l'institut CNUCE

(Pise, Italie). Le modèle d'York [DUKE & HARRISON 93] est une variante modulaire du modèle mathématique PIE [DIX & RUNCIMAN 85]. Son but est de fournir un cadre pour une spécification structurée des systèmes interactifs à base de notations orientées modèle comme Z ou VDM. L'approche de CNUCE [PATERNÒ & FACONTI 92], inspirée des travaux sur les modèles graphiques de référence comme GKS [ISO 85], est plus constructive. Ce modèle est utilisé conjointement avec LOTOS [ISO 87], un langage basé sur l'algèbre des processus.

Nous donnons ici un aperçu conceptuel de ces deux modèles, sans rentrer dans les détails liés aux notations mathématiques.

Les interacteurs d'York

Dans le modèle d'York [DUKE & HARRISON 93], un interacteur est défini comme un *composant dans la description d'un système interactif qui encapsule un état, les événements qui manipulent cet état, et les moyens par lesquels cet état est rendu perceptible par l'utilisateur*.

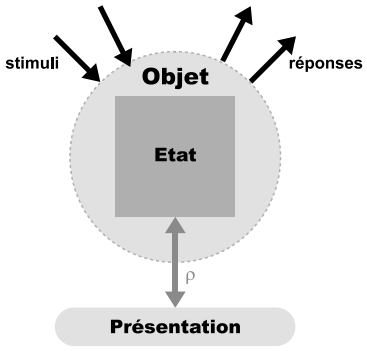


FIG. 2.12 – Schéma général d'un interacteur d'York.

Le schéma général d'un interacteur d'York est représenté sur la figure 2.12. Un interacteur d'York consiste en un *objet* comportant un état et communiquant avec son environnement par des événements de type *stimulus* ou *réponse*. Un interacteur possède également une *présentation*, qui reflète l'état de l'objet de façon perceptible par l'utilisateur. La *relation de rendu* ρ spécifie les présentations possibles pour un état (et éventuellement un historique) donné de l'objet.

Dans une version raffinée de ce modèle, l'élément *présentation* est explicité par un nouvel objet apparié à l'objet principal, et dont l'état interne représente les caractéristiques d'affichage perceptibles par l'utilisateur.

La figure 2.13 page suivante montre comment un comportement simple de système interactif peut être décrit par des combinaisons d'interacteurs. Dans cet exemple, des icônes (disques, fichiers, répertoires) peuvent être manipulées par une souris à un bouton représenté par un curseur. Une icône peut être sélectionnée ou désélectionnée en cliquant à son emplacement, ce qui a pour effet de changer son apparence. Enfin, les icônes sélectionnées peuvent être déplacées en cliquant sur la commande "déplacer" dans un menu, puis en bougeant la souris.

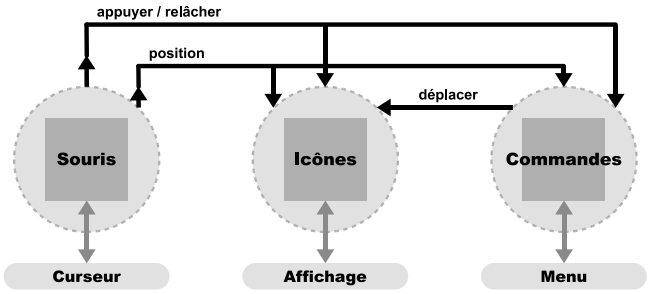


FIG. 2.13 – Un mécanisme simple de sélection et de déplacement d'icônes décrit avec les interacteurs d'York.

Les interacteurs de CNUCE

Le modèle de CNUCE [PATERNÒ & FACONTI 92], plus détaillé que le précédent, décrit en plus la structure et le comportement de base d'un interacteur, distingue plusieurs types et sources d'événements, et introduit la notion de niveau d'abstraction.

Les auteurs de ce modèle définissent un interacteur comme une *entité d'un système interactif capable de réagir à des stimuli externes, en traduisant des données d'un haut niveau d'abstraction vers un niveau plus bas d'abstraction et vice-versa*. Un système interactif est ainsi décrit comme un graphe d'interacteurs communiquant entre eux. Au plus bas niveau, ils communiquent avec l'utilisateur et au plus haut niveau, ils communiquent avec l'application.

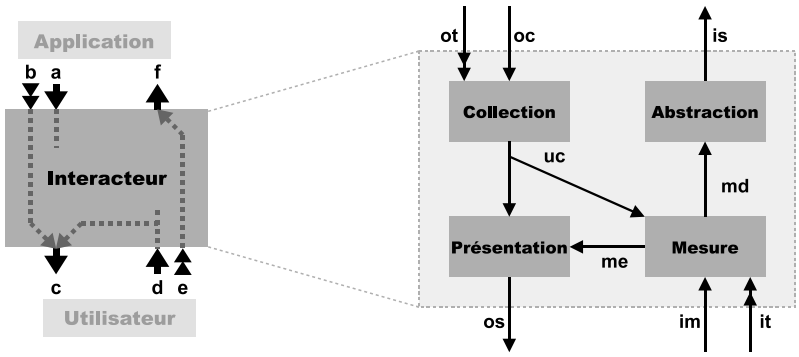


FIG. 2.14 – Le schéma simplifié d'un interacteur de CNUCE (à gauche), et sa structure interne (à droite). Le schéma de droite reproduit les noms de flux employés dans le formalisme original.

Vu de l'extérieur, un interacteur peut (figure 2.14, schéma de gauche) :

- Recevoir et accumuler des informations du côté application (a)
- Recevoir du côté application un signal déclencheur (b) provoquant une émission d'informations du côté utilisateur (c)
- Recevoir et accumuler des informations du côté utilisateur (d) et émettre un retour du côté utilisateur (c)
- Recevoir du côté utilisateur un signal déclencheur (e) provoquant du côté application une émis-

sion des informations accumulées (f).

Vu de l'intérieur, un interacteur est structuré en quatre composants qui communiquent entre eux (figure 2.14 page précédente, schéma de droite) : la *collection* maintient les informations relatives au modèle abstrait de l'interacteur. Lorsque cet élément est déclenché, il transmet ces informations à la *présentation* qui met à jour les éléments visibles par l'utilisateur. D'autre part, la mesure reçoit et accumule les informations provenant de l'utilisateur. Lorsqu'elle est déclenchée, elle les transmet à l'*abstraction* qui les convertit en données abstraites manipulables par l'application. La *mesure* peut également se servir d'informations provenant de la collection.

Notons que l'élément *collection* peut être assimilé à l'*état* de l'interacteur d'York, et que la présentation joue à peu près le même rôle dans les deux modèles. La mesure et l'abstraction n'ont pas d'équivalent dans le modèle d'York, et font implicitement partie de l'état.

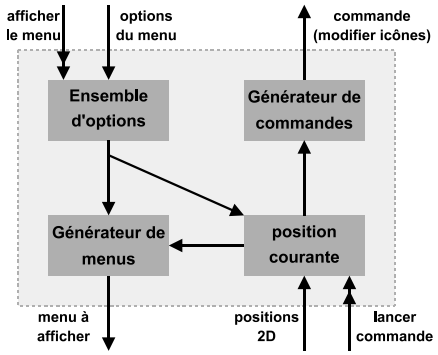


FIG. 2.15 – Vue interne de l'interacteur *Commandes* dans le modèle de CNUCE.

L'exemple de la section précédente peut être décrit avec trois interacteurs, de manière très analogue au modèle d'York. La figure 2.15 reproduit uniquement l'interacteur *Commandes*. Les entrées du côté utilisateur (en bas à droite) sont connectées à l'interacteur *Souris*, et la sortie du côté application (en haut à droite) est connectée à l'interacteur *Icônes*. Les autres connexions sont laissées implicites. Lorsqu'il est déclenché par un clic de souris, l'interacteur compare la position courante stockée dans la *mesure* à la structure stockée dans la *collection*, et selon l'option activée dans le menu, génère l'événement approprié interprétable par les autres interacteurs.

2.3.3 Conclusion

Dans cette section, nous avons donné un bref aperçu de deux approches formelles employées pour décrire les interfaces utilisateur, à savoir les systèmes de transition et les interacteurs formels.

Les systèmes de transition sont particulièrement bien adaptés à la description des comportements de type *contrôle* dans les applications interactives. Ils ont été employés pour décrire le niveau dialogue de l'interaction, le comportement des widgets, ou encore la production d'événements synthétiques. Le modèle des *transducteurs formels* propose un modèle d'entrée transformationnel, et constitue une tentative intéressante de modéliser par rétro-conception le comportement très bas-niveau des dispositifs physiques. Les systèmes de transition ne permettent cependant pas de décrire des parties importantes des interfaces, en particulier les flux de données et l'interaction continue, et c'est pourquoi ils sont fréquemment employés en conjonction avec d'autres formalismes.

Les interacteurs formels sont beaucoup plus structurants pour les interfaces graphiques. Ils décomposent l'interaction en des entités communicantes, à la manière des modèles de référence à agent. Ces interacteurs reçoivent des valeurs en entrée et produisent des sorties. Les interacteurs d'York communiquent entre eux par des stimuli et des réponses, et avec l'utilisateur en maintenant une présentation. Mais la partie entrée de cette communication n'est pas explicite dans le modèle. Le modèle de CNUCE, en plus de décrire une structure et un comportement interne pour les interacteurs, emploie comme les transducteurs formels un modèle d'entrées de type transformationnel basé sur des couches d'abstractions successives.

2.4 Les modèles de dispositifs

Face à une prolifération incontrôlée des dispositifs d'entrée, il est rapidement devenu nécessaire de construire des modèles pour faire face à une telle complexité. Nous distinguons deux grands courants dans les modèles de dispositifs d'entrée : les *modèles logiques*, dont l'objectif est de faciliter l'implémentation d'applications interactives portables, et les *modèles physiques*, dont le but est de mieux comprendre et exploiter la grande variété des dispositifs existants.

Les modèles logiques sont intimement liés à l'histoire de la standardisation de l'informatique graphique. Nous les décrivons dans un premier temps. Les modèles physiques, qui consistent principalement en deux taxinomies, seront décrites par la suite.

2.4.1 Les modèles logiques

Les dispositifs logiques de GKS et PHIGS

Il y a trente ans, les applications graphiques interactives étaient encore dédiées à un matériel spécifique et nécessitaient énormément de programmation bas-niveau. Sentant qu'il devenait nécessaire de disposer d'un standard pour ces machines, la communauté de l'informatique graphique jeta les premières bases d'une API standard avec Core [DUNN & HERZOG 77], qui aboutit au standard international GKS [ECKERT *et al.* 79, ISO 85]. Par la suite, deux extensions 3D de GKS furent proposées : 3D-GKS [KANSY 85] et PHIGS [HEWITT 84] (Programmer's Hierarchical Interactive Graphics System), auquel succéda PHIGS+ [VAN DAM 88] (figure 2.16).

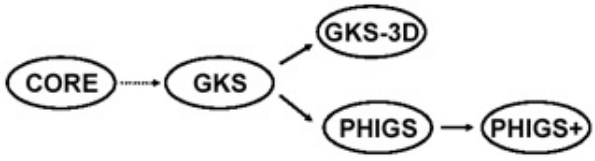


FIG. 2.16 – Historique des standards graphiques.

L'objectif des standards graphiques est l'indépendance du code vis-à-vis du matériel. En termes de sorties, cette indépendance se fait par la définition d'une API graphique générique. Pour les entrées, le standard GKS introduit le modèle des *dispositifs d'entrée logiques*, qui sera repris dans PHIGS et PHIGS+.

Le modèle des dispositifs logiques identifie six (ou cinq, selon la spécification) *classes d'entrées*. Ces classes représentent des dispositifs physiques génériques qui diffèrent par le type de données qu'elles peuvent fournir à l'application :

Locator : position de type (x,y).

Stroke : série de points de type $(x,y)^n$.

Valuator : valeur réelle ou entière bornée.

Choice : entier représentant une sélection parmi un ensemble d'alternatives.

Pick : identifiant d'un objet présent à l'écran.

String : chaîne de caractères.

Sur un système donné, chacun de ces dispositifs logiques est implémenté par un dispositif physique et une technique d'interaction particulières. Par exemple, une valeur de type *Choice* peut être spécifiée par un tableau de boutons ou par un menu graphique. Ce paradigme garantit la portabilité des applications et simplifie la tâche du programmeur, dans l'hypothèse où celui-ci ne s'intéresse pas à la façon dont les données utilisateur sont obtenues.

Les tâches d'interaction de Foley

Après avoir contribué au standard GKS, James Foley a introduit la notion de *tâches d'interaction* génériques [FOLEY *et al.* 84]. Plus ou moins calquées sur les dispositifs logiques, ces tâches d'interaction mettent l'accent sur les intentions de l'utilisateur plutôt que sur des types de données. Elles sont au nombre de six :

Select : sélection d'un objet.

Position : positionner un objet sur 1, 2, 3 dimensions ou plus.

Orient : orienter un objet sur 1, 2, 3 dimensions ou plus.

Ink : dessiner une ligne.

Text : saisir un texte.

Value : spécifier une valeur scalaire.

Foley énumère alors, pour chaque tâche d'interaction, les techniques d'interaction et les dispositifs physiques permettant de la réaliser. La figure 2.17 page suivante illustre les relations possibles entre trois tâches (les racines de l'arbre) et les dispositifs physiques (les feuilles de l'arbre). Cet arbre peut être lu comme une taxinomie de dispositifs, bien que ceux-ci apparaissent plusieurs fois dans la représentation.

2.4.2 Les modèles physiques

La taxinomie de Buxton

Selon William Buxton, les abstractions fournies par les modèles logiques sont utiles pour le programmeur, mais vont à l'encontre des considérations d'utilisabilité [BUXTON 83]. En particulier,

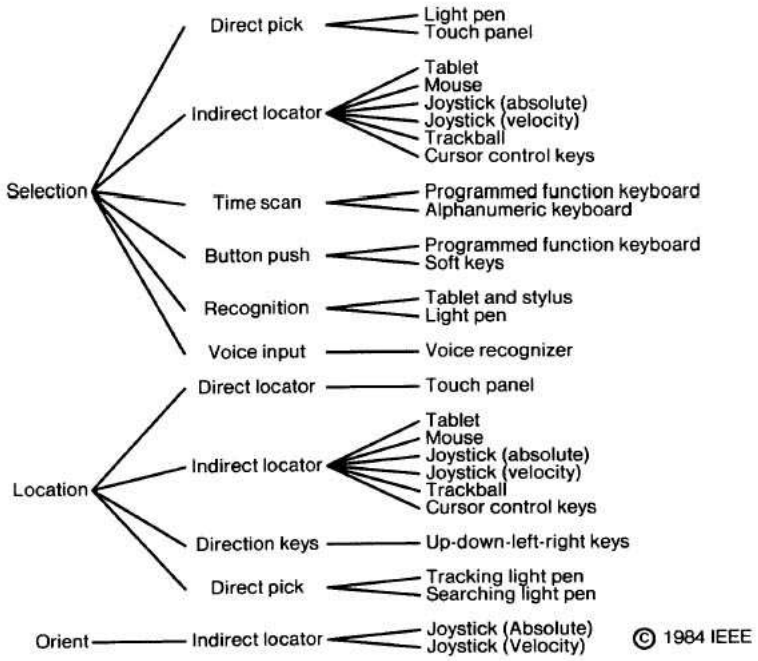


FIG. 2.17 – La taxinomie de Foley [FOLEY *et al.* 84].

elles présupposent que deux dispositifs peuvent être totalement interchangeables (voir la section 1.2.1 page 7 à ce sujet). Or les aspects physiques des interfaces, identifiés par Buxton comme le « niveau pragmatique » de l'interaction, ont des effets extrêmement importants sur la perception des systèmes par l'utilisateur.

Buxton recommande d'inclure les *caractéristiques pragmatiques* des dispositifs d'entrée dans les spécifications de dispositifs génériques. Il propose un ensemble de caractéristiques pertinentes, avec lequel il construit une taxinomie des dispositifs d'entrée. Cette taxinomie est reproduite sur la figure 2.18 page ci-contre.

Les deux principaux axes de la taxinomie sont les suivants :

Nombre de dimensions : Les dispositifs à *une dimension* comprennent les dispositifs de type potentiomètre rotatif ou linéaire. Les dispositifs à *deux dimensions* comprennent l'ensemble des dispositifs de pointage et des manettes. Les dispositifs à *trois dimensions* incluent les manettes 3D et trackballs 3D.

Propriété captée : Parmi les dispositifs précédents, certains captent une *position* (tous les dispositifs isotoniques), d'autres un *déplacement* (souris, dispositifs élastiques), et d'autres enfin une *pression* (dispositifs isométriques).

Buxton suggère également de prendre en compte les aspects continu/discret ainsi que l'agent de contrôle (main, pied, voix, …), seuls les dispositifs manuels et continus étant représentés dans sa taxinomie. Une liste très complète des caractéristiques pragmatiques à prendre en compte a été plus tard fournie par Lipscomb et Pique [LIPSCOMB & PIQUE 93].

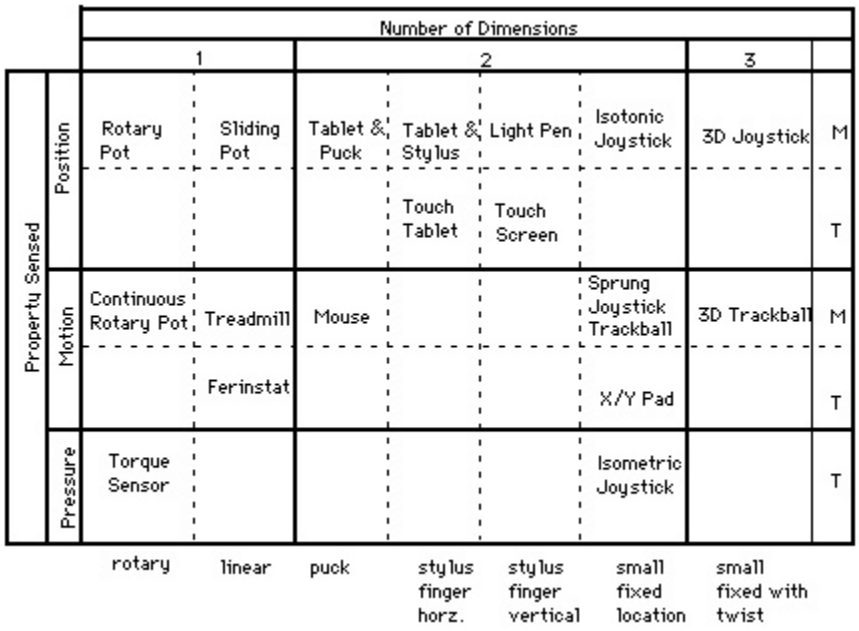


FIG. 2.18 – La taxinomie de Buxton [BUXTON 83].

L'espace de conception de Card, Mackinlay et Robertson

En s'inspirant de la taxinomie précédente, Card, Mackinlay et Robertson [CARD *et al.* 90, CARD *et al.* 91] ont proposé *un espace de conception* pour les dispositifs d'entrée, donnant lieu à une taxinomie générale et détaillée des dispositifs existants qui peut également servir de support à la création de nouveaux dispositifs.

Partant du principe qu'un dispositif d'entrée traduit des propriétés physiques du monde en des paramètres logiques d'une application, Card définit un dispositif d'entrée par un n-uplet $\langle M, In, S, R, Out, W \rangle$ où :

- **M** est l'*opérateur de manipulation* qui décrit le type de propriété captée : position ou force, absolu ou relatif, linéaire ou angulaire.
- **In** est le *domaine d'entrée*, c'est-à-dire les valeurs prises par la grandeur physique captée.
- **S** est l'*état courant* du dispositif.
- **R** est la *fonction de résolution*, qui à chaque valeur du domaine d'entrée associe une valeur du le domaine de sortie.
- **Out** est le *domaine de sortie* qui décrit les valeurs logiques possibles.
- **W** est un ensemble de *propriétés* qui décrivent des aspects fonctionnels supplémentaires du dispositif.

Trois opérateurs décrivent les différentes manières de composer ces dispositifs atomiques pour former des dispositifs plus complexes (figure 2.19 page suivante) :

- **L'union** (merge) consiste à composer deux dispositifs afin que le domaine d'entrée résultat soit le produit cartésien des deux domaines d'entrée sources. Par exemple, une souris peut être considérée comme l'union de deux potentiomètres linéaires orthogonaux.

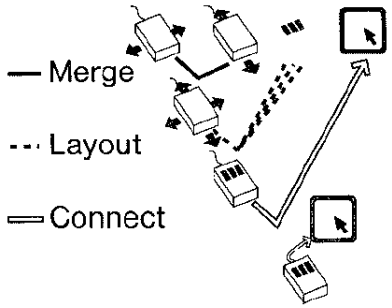


FIG. 2.19 – Construction d'une souris à l'aide des trois opérateurs de composition [CARD *et al.* 91].

– **L'agencement** (layout) consiste à composer spatialement deux dispositifs. Un exemple est l'agencement spatial des axes (x,y) et des trois boutons d'une souris.
– **La connexion** (connect) consiste à composer deux dispositifs afin que le domaine de sortie du premier soit relié au domaine d'entrée du second. Un exemple est la connexion d'une souris à un pointeur (dispositif logiciel).

L'espace de conception de Card, illustré sur la figure 2.20 page ci-contre, met en évidence les principales propriétés de chaque dispositif d'entrée ainsi que sa décomposition en dispositifs atomiques. L'opérateur de manipulation se retrouve en abscisse (linéaire/angulaire et direction de l'axe) et en ordonnée (position absolue ou relative, force absolue ou relative). Sur ce schéma ont été placés un grand nombre de dispositifs d'entrée, dont ceux auparavant décrits par Foley et par Buxton.

2.4.3 Conclusion

Si les normes GKS et PHIGS sont longtemps restées une référence dans le domaine de l'informatique graphique, leur modèle d'entrée s'est très rapidement révélé dépassé par les interfaces modernes [MYERS 90]. Le modèle des dispositifs logiques est adapté à l'interaction modale avec les applications graphiques type CAO de l'époque, mais non à la manipulation directe moderne. Les tâches d'interaction de Foley sont également ad-hoc.

Très tôt pourtant, les taxinomies de dispositifs de Buxton et de Card ont mis en évidence le caractère *ad-hoc* des modèles logiques. Ces taxinomies, extrêmement intéressantes du point de vue théorique et pédagogique, et abondamment référencées dans la littérature scientifique, ne sont malheureusement pas de bons modèles d'implémentation et leurs applications logicielles sont pratiquement inexistantes.

Les standards GKS et PHIGS, focalisés sur l'aspect graphique, n'ont pas évolué du point de vue des entrées. Aujourd'hui, ils ont été remplacés par des standards industriels *de facto* comme OpenGL [CARSON 97], qui sont dédiés à l'aspect graphique et ne traitent pas le problème de l'interaction en entrée.

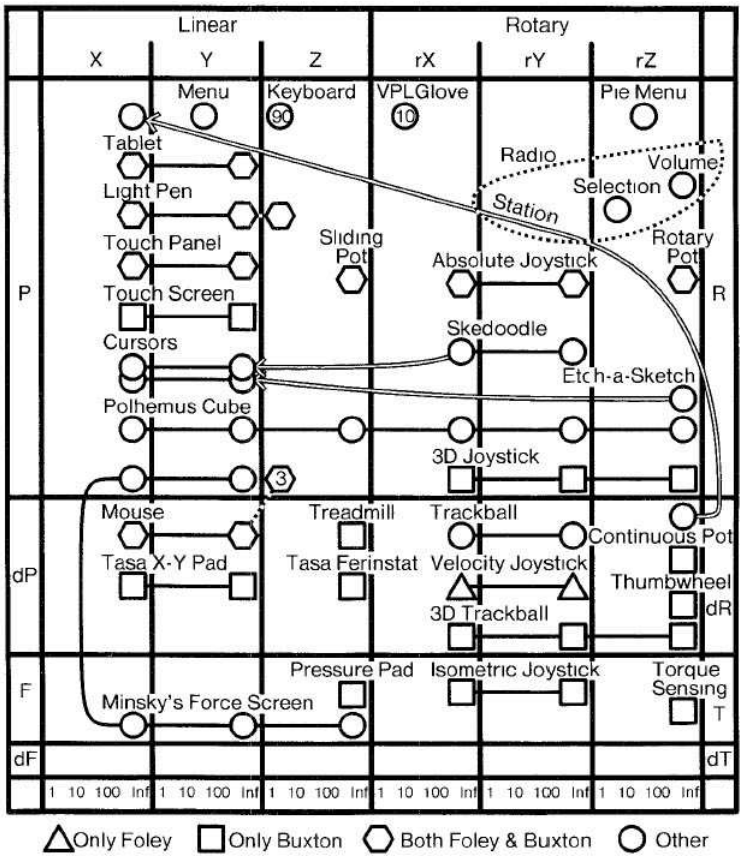


FIG. 2.20 – L'espace de conception de Card [CARD *et al.* 91].

2.5 Les modèles d'interaction

« Un modèle d'interaction est un ensemble de principes, de règles et de propriétés qui guident la conception d'une interface » [BEAUDOUIN-LAFON 97]. Un modèle d'interaction détaille tous les aspects d'un paradigme d'interaction du point de vue utilisateur et inspire des principes architecturaux exploitables par le développeur. Dans cette section, nous rappelons les principes généraux du modèle de la *manipulation directe*, qui a en partie inspiré les interfaces graphiques modernes que nous utilisons actuellement. Puis nous décrivons une extension et une généralisation de ce modèle, l'*interaction instrumentale*, qui jette les bases des applications post-WIMP.

2.5.1 La manipulation directe

La transition entre les interfaces à ligne de commande du type *UNIX* et les interfaces graphiques actuelles a été initiée dès les années 60 avec des applications de démonstration comme *Sketchpad* et *Pygmalion*, pour finalement se concrétiser vers les années 80 avec le système *Xerox Star* qui intègre toutes les caractéristiques de nos interfaces actuelles [MYERS 98A]. Peu après, Ben Shneiderman [SHNEIDERMAN 83] invente le terme de « manipulation directe » et formule le modèle d'interaction que nous décrivons ici.

La *manipulation directe* [SHNEIDERMAN 83, SHNEIDERMAN 98] décrit des systèmes interactifs ayant les caractéristiques suivantes :

1. La visibilité des objets d'intérêt et des actions possibles,
2. Des actions rapides, réversibles, incrémentales,
3. Le remplacement de la syntaxe complexe des langages de commande par la manipulation directe de l'objet d'intérêt.

La manipulation directe repose sur une métaphore du monde réel, dans lequel nous manipulons directement les objets. Shneiderman décrit quelques exemples d'interfaces exploitant la manipulation directe, dont les traitements de texte WYSIWYG du type Microsoft Word, les tableurs graphiques, les jeux vidéo, les logiciels de CAO [SHNEIDERMAN 98]. Les avantages de ce paradigme sont nombreux : apprentissage rapide et productivité accrue, messages d'erreurs moins nécessaires, réduction de l'anxiété due à un système compréhensible et des actions réversibles.

Hutchins et al. [HUTCHINS *et al.* 86] clarifient le concept de manipulation directe du point de vue cognitif, et évoquent un « sentiment d'engagement dans un monde d'objets plutôt que l'impression de communiquer par un intermédiaire ». Ils définissent une notion de « directitude » mesurée par le *fossé d'exécution* et le *fossé d'évaluation*. Le premier réfère à la distance entre ce que l'idée que l'utilisateur se fait de la tâche et la façon dont elle est représentée par le système. Le second évoque la distance entre le comportement du système et les objectifs de l'utilisateur. L'objectif principal de la manipulation directe est de combler ces fossés.

Nos interfaces actuelles s'inspirent des principes de la manipulation directe. Elles reposent cependant sur un paradigme qui n'est qu'une version appauvrie de la manipulation directe (voir la section 1.1.1 page 3). Les widgets qui composent les interfaces WIMP, en particulier, sont des objets directement manipulables mais qui ne servent que d'intermédiaires aux réels objets d'intérêt (figure 2.21 page ci-contre).

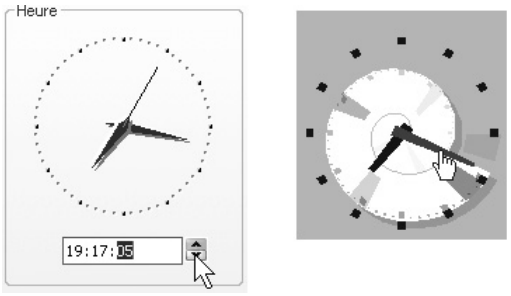


FIG. 2.21 – À gauche : réglage de l'heure dans Windows XP avec des techniques WIMP. À droite : manipulation directe de l'heure dans SpiraClock [DRAGICEVIC & HUOT 02].

2.5.2 L'interaction instrumentale

Le paradigme d'interaction instrumentale [BEAUDOUIN-LAFON 97, BEAUDOUIN-LAFON 00] s'inspire de notre expérience interactive avec le monde physique, qui est gouvernée par l'utilisation d'*outils*. Des outils comme un pinceau ou une perceuse, ou même un interrupteur, sont des objets intermédiaires (*instruments*) que nous utilisons pour agir sur d'autres objets (*objets d'intérêt*).

Dans cette section, nous présentons le modèle de l'interaction instrumentale tel qu'il a été décrit par [BEAUDOUIN-LAFON 97].

Instruments d'interaction

Les données manipulées par une application peuvent être représentées par des entités appelées *objets du domaine*, eux-mêmes décrits par un ensemble d'*attributs* : ainsi, une sphère dans un modeleur 3D est définie par des attributs simples comme sa position et sa taille, et d'autres plus complexes comme sa couleur et sa texture.

L'interaction avec une application a pour finalité la manipulation des objets du domaine. Cette manipulation peut prendre deux formes : soit l'utilisateur agit sur les attributs de ces objets, soit il manipule ces objets comme un tout, par exemple en les copiant ou en les effaçant.

Les manipulations s'effectuent par l'intermédiaire d'*instruments d'interaction*, qui sont les médiateurs entre l'utilisateur et les objets du domaine. Une barre de défilement est un exemple d'instrument qui opère sur un document (l'objet du domaine) en modifiant sa partie visible (un attribut).

Les instruments séparent l'interaction en deux couches, une du côté utilisateur, et une du côté application (figure 2.22 page suivante). Les communications du côté utilisateur consistent en *actions* physiques de l'utilisateur et *réactions* de l'instrument. Les messages du côté application consistent en *commandes* envoyées à l'objet et *réponses* de celui-ci, que l'instrument peut transformer en *retour* vers l'utilisateur.

Le comportement typique d'un instrument est le suivant :

– L'utilisateur agit sur l'instrument, qui traduit ses *actions* en *commandes* qui altèrent les objets du domaine. Exemple : l'utilisateur clique sur un des boutons fléchés de la barre de défilement, ce qui a pour effet d'envoyer une commande de défilement au document (cette commande est

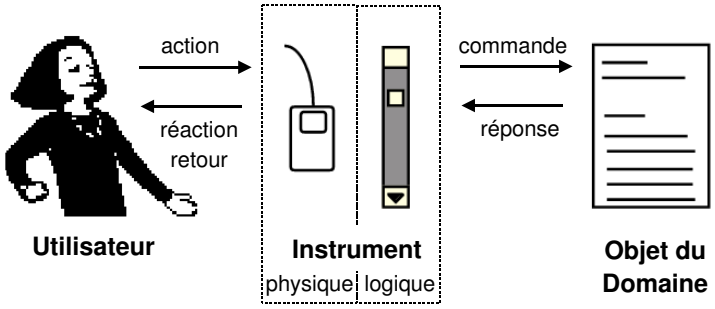


FIG. 2.22 – Schéma d'un instrument.

réémise à intervalles tant que le bouton reste appuyé).
- L'instrument produit des *réactions* qui permettent à l'utilisateur de contrôler son action sur l'instrument. Exemple : le bouton de la barre de défilement s'enfonce dès que l'utilisateur clique dessus.
- L'instrument produit également un *retour* donnant une idée de l'effet des commandes sur les objets du domaine. Exemple : la barre de défilement met à jour la position du pouce, indiquant la nouvelle position dans le document. Un deuxième niveau de retour réaffiche la partie visible du document courant.

Activation spatiale et temporelle

Un instrument comporte une partie physique (le dispositif d'entrée) et une partie logique (sa représentation logicielle et visuelle) (voir figure 2.22). Dans la grande majorité des applications interactives, de nombreux instruments logiques se partagent le même dispositif physique. Les ambiguïtés sont alors résolues par des mécanismes d'*activation* (également appelés mécanismes de *multiplexage* [BUXTON 86B]. Un instrument est dit *activé* lorsqu'il est associé à un dispositif physique pour être contrôlé par l'utilisateur.

Les mécanismes d'activation sont de deux sortes : l'*activation spatiale* est liée à la présence d'un pointeur à l'intérieur d'une zone visuelle, alors que l'*activation temporelle* est liée à des changements de mode demandant des actions explicites (par exemple, le fait de cliquer sur un bouton dans une barre d'outils). L'activation spatiale est plus directe et plus rapide que l'activation temporelle. Cependant, l'activation spatiale requiert des instruments visibles en permanence à l'écran (widgets) qui occupent de la place.

Dans tous les cas, les coûts d'activation sont non négligeables lorsque le nombre d'instruments devient important. Ces coûts peuvent être réduits par des *associations implicites* entre dispositifs et instruments, comme par exemple l'association entre la molette de la souris et la barre de défilement. L'utilisation d'associations implicites passe par la multiplication des dispositifs physiques.

Notons enfin que la manipulation d'un instrument n'implique pas seulement son association avec un dispositif physique, mais également son association avec un objet du domaine. L'entité qui, à un instant donné, fait l'objet d'une manipulation instrumentale est appelé *objet d'intérêt* ; c'est l'objet sur lequel l'utilisateur focalise temporairement son attention. Cette notion fondamentale de la manipulation directe est également mise en avant dans la manipulation instrumentale.

Méta-instruments

Dans le monde réel, il est courant que l'attention se focalise temporairement sur l'instrument lui-même : un crayon peut par exemple être affûté par un taille-crayon, qui à son tour peut être resserré par un tournevis. De façon analogue, un instrument peut être vu comme un objet du domaine, qui peut être contrôlé par un autre instrument.

Dans les applications interactives, les menus et les barres d'outils sont des exemples de *méta-instruments* qui servent à activer d'autres instruments. Les méta-instruments peuvent également s'avérer utiles pour organiser des instruments dans un espace de travail ou spécialiser des instruments pour des tâches spécifiques.

La réification

La *réification* est le processus qui consiste à transformer un concept en objet. L'utilisation de méta-instruments est un exemple de réification d'instruments. Les styles, qui sont des attributs partagés entre plusieurs objets, peuvent également faire l'objet d'une réification et être manipulés par des instruments. En multipliant le nombre et la nature des objets manipulables dans une application, la réification ouvre la voie à de nouvelles possibilités d'interaction, tout en conservant un style uniforme et cohérent calqué sur la manipulation instrumentale.

Propriétés des instruments

Le *degré d'indirection*, le *degré d'intégration* et le *degré de compatibilité* sont trois propriétés essentielles des instruments. Ces propriétés permettent comparer entre eux plusieurs instruments effectuant des tâches similaires. Elles ont également servi de base à une taxinomie des techniques d'interaction WIMP et post-WIMP.

- Le *degré d'indirection* traduit le décalage spatial et temporel entre un instrument et l'objet sur lequel il agit. Certains instruments comme les poignées de redimensionnement[1] sont placés directement sur l'objet d'intérêt. D'autres, comme les boîtes de dialogue, peuvent se trouver éloignés de l'objet qu'elles manipulent. Le système de validation des boîtes de dialogue ajoute également au décalage spatial un décalage temporel.
- Le *degré d'intégration* est le rapport entre le nombre de dimensions physiques et le nombre de dimensions logiques qui sont manipulées lors de l'utilisation de l'instrument. Pour une barre de défilement (1D) manipulée par une souris (2D), il est de 1/2. Pour un objet 3D manipulé par le même dispositif, il est de 3/2. Cette propriété est directement liée à la notion de *tâches intégrales* introduite par Robert Jacob [JACOB *et al.* 94].
- Le *degré de compatibilité* traduit la similarité entre les manipulations effectuées sur l'instrument et leurs effets sur l'objet d'intérêt. À titre d'exemple, la barre de défilement possède un degré de compatibilité plus faible que le glisser-déposer, car le défilement du document est inversé par rapport à la manipulation.

[1] Il s'agit habituellement de neuf carrés pleins disposés autour de l'objet qui apparaissent lorsque cet objet est sélectionné, afin de permettre de le redimensionner.

2.5.3 Conclusion

Le modèle de la manipulation directe marque une transition importante entre les interfaces archaïques à lignes de commande et les interfaces graphiques actuelles, où les objets sont représentés à l'écran et manipulés avec une souris. Ce modèle est cependant trop abstrait pour garantir des interfaces où les fossés d'exécution et d'évaluation ont été réduits à leur strict minimum. La preuve en est de nos interfaces actuelles, où les objets d'intérêt sont presque toujours manipulés de façon indirecte.

Le modèle de l'interaction instrumentale affine le modèle de la manipulation directe en intégrant les paradigmes Post-WIMP qui ont été régulièrement proposés pour corriger les faiblesses de nos interfaces modernes. Aussi, la métaphore précédente qui décrit la manipulation d'objets de notre vie quotidienne a été étendue à l'utilisation d'outils (instruments) comme objets médiateurs. Tout en proposant l'exploration de nouveaux paradigmes comme les méta-instruments (des instruments qui agissent sur d'autres instruments), ce modèle guide la conception d'interfaces réellement directes. Ainsi, les trois propriétés essentielles que sont le degré d'indirection, le degré d'intégration et le degré de compatibilité expliquent notamment pourquoi les widgets sont des instruments peu efficaces et montrent l'intérêt de multiplier les dispositifs physiques et de préserver au maximum les compatibilités entre le dispositif et la tâche.

2.6 Les outils de développement

Une application interactive a essentiellement besoin d'animer des formes géométriques à l'écran et de lire les données en provenance des dispositifs d'entrée. Le développement d'une application complète avec des librairies graphiques et des librairies d'entrée bas-niveau constitue cependant un travail titanesque. C'est pourquoi les outils de type *Xlib* ont rapidement cédé la place à des librairies de plus haut niveau comme la *X Toolkit* [McCormack & Asente 88]. Ces *boîtes à outils graphiques* ou *boîtes à outils d'interaction* proposent des jeux d'objets interactifs réutilisables (*widgets*) et un mécanisme de gestion des entrées basée sur la notion d'*événements*. La plupart du temps, l'implémentation de la partie interactive d'une application se résume alors à l'instanciation et au positionnement déclaratif de widgets, dont la manipulation est gérée automatiquement par des mécanismes d'*aiguillage événementiel* et des *comportements* prédéfinis au sein des widgets. Ces mécanismes seront décrits plus en détail par la suite.

Si les appréciations divergent quant à la facilité d'utilisation des boîtes à outils graphiques, tous les développeurs s'accordent à dire qu'elles sont difficiles à étendre (du moins, ceux qui s'y sont essayé) [Accot *et al.* 98]. Cette difficulté est encore bien plus marquée du point de vue de l'interaction en entrée. En effet, fidèles au modèle de l'interaction standard, les boîtes à outils sont toutes câblées pour une utilisation exclusive et stéréotypée de la souris et du clavier (voir la section 1.1.1 page 3 pour une définition de l'interaction standard et la section 1.4.1 page 29 pour une discussion sur ses faiblesses). Les principales difficultés proviennent de ce qu'elles prennent en charge un ensemble limité et statique de types d'événements (souris et clavier), emploient des mécanismes d'aiguillage complexes dont l'architecture est floue, et entremêlent les aspects graphiques et comportementaux.

Depuis les premières boîtes à outils, de grands progrès ont été faits du point de vue graphique : des abstractions simplifient grandement la gestion de l'affichage et des effets visuels toujours plus sophistiqués sont pris en compte. Ces avancées ont une certaine influence sur l'interaction en entrée :

le dessin structuré[2] permet d'implémenter plus facilement des techniques de manipulation directe [BEAUDOUIN-LAFON *et al.* 90] et la prise en charge d'effets visuels avancés tels que la transparence ou les déformations encouragent l'innovation dans les techniques d'interaction [ROUSSEL 02]. Malgré tout, les paradigmes d'interaction Post-WIMP avancés nécessitent pour être décrits un remaniement non négligeable voire complet du modèle standard de gestion des entrées. C'est encore plus le cas lorsque l'on désire obtenir une certaine configurabilité de l'interaction. Or même les outils modernes comme Java Swing [ECKSTEIN & LOY 02] reposent sur un modèle vieux de plusieurs décennies.

Dans cette section, nous donnons un large aperçu des boîtes à outils qui remettent en cause de façon significative le modèle d'entrée rigide conventionnel. Nous commençons par décrire les outils qui modélisent les paradigmes d'interaction standard de façon claire et extensible, et que nous nommons *boîtes à outils WIMP avancées*. Puis, nous décrivons les outils qui reposent sur des architectures dédiées à d'autres paradigmes d'interaction, et que nous nommons *boîtes à outils Post-WIMP spécialisées*.

2.6.1 Les boîtes à outils WIMP avancées

L'objectif des boîtes à outils WIMP avancées est de fournir de bons modèles et de bons outils permettant d'une part de faciliter la construction d'interfaces WIMP ou à manipulation directe conventionnelles, d'autre part de favoriser la description de techniques plus spécifiques ou moins conventionnelles. Comme preuve d'extensibilité, ces boîtes à outils prennent en charge un ensemble minimal de techniques d'interaction non-standard.

Les deux contributions les plus importantes dans ce domaine sont les boîtes à outils *Surbactic* [HUDSON & SMITH 96A] et *Garnet/Amulet* [MYERS 90, MYERS *et al.* 97]. Nous les présentons ici, et détaillons les mécanismes qu'elles emploient pour la gestion des entrées. Nous analysons ensuite leurs apports respectifs, ainsi que leurs limites.

Subarctic Toolkit

Subarctic Toolkit [HUDSON & SMITH 96A] est une boîte à outils *Java* qui prend en charge des effets visuels avancés et les animations [HUDSON & STASKO 93], et possède un moteur de contraintes[3] pour la gestion de l'organisation spatiale des widgets [HUDSON & SMITH 96B]. Le modèle d'entrée de cette boîte à outils a été introduit dans *Artkit* [TYSON R. *et al.* 90], le prédécesseur de *Subarctic* développé en C^{++}. L'objectif principal de ce modèle est de décrire les mécanismes d'aiguillage employés par les boîtes à outils traditionnelles de façon claire et extensible, pour pouvoir y intégrer ensuite de nouveaux mécanismes. En plus des interactions habituelles, *Subarctic* prend en charge trois paradigmes d'interaction non conventionnels, à savoir l'*interaction gestuelle* [TYSON R. *et al.* 90], les *champs de gravité*[4] [TYSON R. *et al.* 90, HUDSON 90], et les *lentilles sémantiques*[5] [HUDSON *et al.* 97].

[2] Dans le *modèle de dessin structuré*, les appels à des routines graphiques sont remplacés par la manipulation d'une *liste d'affichage* dont le rendu est pris en charge par le système.

[3] Les paradigmes de *programmation par contraintes* et de *flot de données*, auxquels nous ferons parfois référence dans cette partie, seront décrits plus en détail dans la partie sur les langages visuels (section 2.7.1 page 79)

[4] Également appelée *snap-dragging*, il s'agit d'une technique de cliquer-glisser dans laquelle l'objet manipulé est attiré vers des positions-clés [BIER & STONE 86]. Elle permet à la fois de faciliter la manipulation et de prévenir les erreurs.

[5] Les lentilles sémantiques sont des formes flottantes qui se comportent comme des filtres, qui effectuent des transformations graphiques sophistiquées (loupe, par exemple) ou exposent des représentations graphiques alternatives. Les filtres de

Dans toute boîte à outils, l'*aiguillage* des entrées consiste à *distribuer les événements d'entrée aux objets appropriés dans l'arbre des widgets*. La façon dont *Subarctic* distribue ces entrées est synthétisée sur la figure 2.23. Les rectangles représentent des objets (au sens programmation par objets) qui sont de type *Politique d'Aiguillage* ou *Agent d'Aiguillage*.

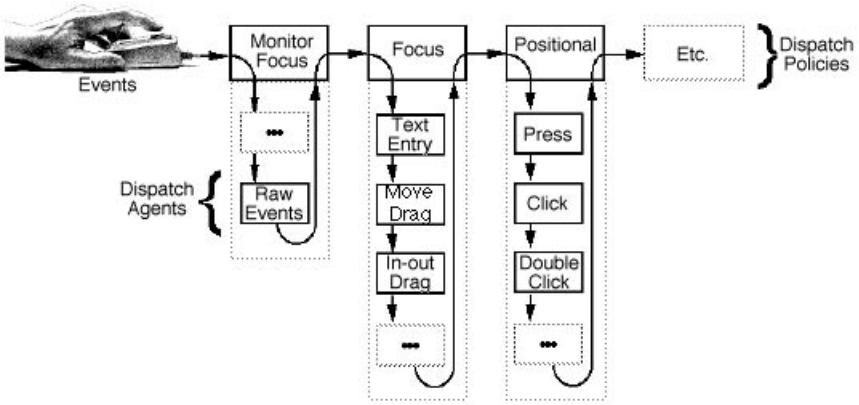


FIG. 2.23 – La gestion des entrées dans Subarctic [HUDSON & SMITH 96A].

Les Politiques d'Aiguillage (*Dispatch Policies*) : Les politiques d'aiguillage décrivent les principales stratégies de distribution des événements. Ces objets sont disposés horizontalement en haut de la figure 2.23, et nous les énumérons de droite à gauche :

- **l'Aiguillage Positionnel** (*Positional*) consiste à envoyer les événements positionnels aux widgets qui se trouvent sous le pointeur, dans l'ordre inverse de leur affichage.
- **l'Aiguillage par Focus** (*Focus*) consiste à transmettre certains événements à un widget donné, indépendamment de la position du pointeur. Les événements en provenance du clavier sont envoyés au widget qui détient le focus clavier. Les événements de type *drag* sont transmis au widget sur lequel a été initié le cliquer-glisser.
- **l'Aiguillage de Contrôle** (*Monitor Focus*) consiste à envoyer les événements bruts aux objets intéressés, à des fins de débogage, par exemple.

Ces trois politiques sont listées sur la figure dans leur ordre de priorité décroissante. Chaque événement passe d'une politique à l'autre jusqu'à ce qu'il trouve un destinataire intéressé, qui *consomme* alors l'événement. Dans l'aiguillage de contrôle cependant, les destinataires interprètent mais ne consomment pas les événements.

Les Agents d'Aiguillage (*Dispatch Agents*) : Chaque politique d'aiguillage est constituée d'un ensemble d'agents d'aiguillage agencés verticalement sur la figure. Un agent d'aiguillage implémente un *aspect donné* de la stratégie d'aiguillage, et se charge de convertir les événements bas-niveau en événements *plus haut-niveau* avant de tenter de les transmettre aux widgets. L'ensemble des événements synthétiques qu'un agent est susceptible de produire constitue un *protocole d'entrée*, et les widgets intéressés par ces événements implémentent ce protocole d'entrée (caractérisé par une interface dans le paradigme de la programmation par objets). Voici quelques exemples d'agents :

- **Press :** Cet agent filtre simplement les événements souris de type *press* et *release*, qu'il transmet aux widgets intéressés qui se trouvent sous le pointeur.

débogage [HUDSON *et al.* 97], qui affichent des informations sur les widgets, en sont un exemple. Nous n'insisterons pas sur ce paradigme, qui constitue davantage une technique de visualisation que d'interaction.

- **Click** et **Double-Click :** À partir des mêmes événements *press* et *release*, ces agents produisent et transmettent des événements synthétiques de type *click* ou *double-click*.
- **Move Drag :** Cet agent produit des événements de type *drag_start*, *drag_feedback* et *drag_end* à partir d'événements souris de type *press*, *move* et *release*. Comme dans la plupart des agents, la production d'événements synthétiques y est régie par un automate à états (se référer à la figure 2.10 page 45 pour celui-ci).

Chaque événement reçu par une politique d'aiguillage est successivement transmis à chacun de ses agents selon leur ordre de priorité, puis à la politique suivante si l'événement n'a toujours pas été consommé.

Le gestionnaire d'entrées de *Subarctic* peut être étendu afin de prendre en charge d'autres techniques d'interaction. De nouveaux agents ou politiques d'aiguillage peuvent être implémentés, éventuellement en dérivant un objet existant, puis insérés à l'endroit voulu dans la liste de priorité. Par exemple, une boîte de dialogue modale peut être implémentée par une politique hybride focus/positionnel qui transmet les événements de façon positionnelle aux enfants d'un widget qui détient le focus. L'interaction gestuelle et les champs de gravité ont tous deux été implémentés par des agents d'aiguillage de type focus.

Pour le premier, un agent *Segmentation* a été dérivé de l'agent *Inking Drag*, lui-même dérivé de l'agent *Move Drag* auquel il ajoutait un retour graphique de type trace. Cet agent transforme les événements souris en séries de segments pour les transmettre à un moteur de reconnaissance. Des types particuliers d'objets, les *zones sensibles*, ont été créés pour servir d'intermédiaires entre l'agent de segmentation et le widget qui possède le focus gestuel. Il s'agit de widgets invisibles qui encapsulent une zone gestuelle et le vocabulaire gestuel associé. Quant à la seconde technique, elle emploie un agent *Snap* qui étend l'agent *Move Drag* en ajoutant à son protocole d'entrée les événements *snap_feedback* et *unsnap_feedback*. Ces événements notifient le widget manipulé de son passage à proximité d'un *site actif* sémantiquement valide. Les sites actifs sont des points qui agissent comme des champs de gravité et qui sont déclarés par les widgets cibles. Il peut s'agir par exemple de cibles de connexion dans un éditeur de diagrammes.

Garnet/Amulet

L'objectif principal de la boîte à outils *Garnet* [MYERS 90] et de son successeur *Amulet* [MYERS *et al.* 97] est de simplifier le développement d'applications hautement interactives en séparant du noyau applicatif les principaux aspects de l'interaction (principalement, la gestion de la présentation, des entrées et du annuler/refaire), et en fournissant des outils et des abstractions adaptés à chacun de ces aspects. *Garnet* est implémenté dans le langage *CommonLisp*, qu'il étend avec un modèle objet à prototypes et un paradigme de programmation par contraintes ; *Amulet* repose sur une infrastructure C^{++}, qu'il instrumentalise de la même manière.

Le principe des *interacteurs* [MYERS 90] introduit dans *Garnet* offre un modèle de haut-niveau pour la gestion des entrées. Ce modèle part de l'observation que dans les boîtes à outils traditionnelles, la programmation du *comportement d'un widget*, c'est-à-dire sa réaction aux événements souris et clavier, est pénible et répétitive : il est notamment fréquent de devoir décrire plusieurs fois les mêmes mécanismes. Dans *Garnet*, ces comportements sont décrits indépendamment des aspects graphiques, encapsulés dans des objets réutilisables appelés *interacteurs*. Suivant une taxonomie très semblable à celle de Foley [FOLEY *et al.* 84], six types d'interacteurs sont fournis pour couvrir la plupart des comportements couramment observés dans les interfaces graphiques :

1. **Menu :** L'interacteur *Menu* couvre les techniques de *sélection* employées dans les menus ou les boutons radio. Il décrit également les comportements des *boutons*.
2. **Move-Grow :** L'interacteur *Move-Grow* décrit les *manipulations géométriques* employées dans la manipulation directe, à savoir la translation et le redimensionnement. Il décrit également le cliquer-glisser avec des widgets de type *barre de défilement*.
3. **New-Point :** L'interacteur *New-Point* couvre les techniques de *création d'objets géométriques* dans les éditeurs graphiques, consistant à spécifier des points-clés.
4. **Angle :** L'interacteur *Angle* décrit les *rotations d'objets* tels que les curseurs de jauges circulaires ou les aiguilles d'horloges analogiques.
5. **Text :** L'interacteur *Text* décrit les principales techniques de saisie employées dans les widgets textuels.
6. **Trace :** L'interacteur *Trace* décrit des techniques faisant usage de séries d'événements positionnels, tels que le dessin à main levée. Rebaptisé *Gesture* dans *Amulet*, cet interacteur prend maintenant en charge les techniques d'interaction gestuelle.

Pour rendre un objet graphique sensible aux événements d'entrée, le programmeur a simplement besoin d'instancier l'interacteur approprié avant de l'associer à l'objet. L'interacteur décrit la partie *C* du modèle *MVC* (voir section 2.2.2 page 40), et vient s'intercaler entre les événements de bas-niveau de type souris et clavier, qu'il interprète, et l'objet graphique, qu'il manipule à travers un protocole. Un seul interacteur peut opérer sur plusieurs objets graphiques, par exemple l'ensemble des objets manipulables dans une zone d'édition graphique. Certains objets graphiques peuvent également comporter plusieurs interacteurs : par exemple, une barre de défilement emploie un interacteur de type *Menu* pour ses deux boutons et un interacteur de type *Move-Grow* pour le pouce.

Un interacteur prend en charge *tous les aspects* de la technique d'interaction qu'il décrit, y compris les feedbacks transitoires comme la création et l'animation d'un objet fantôme dans une technique de cliquer-glisser. À titre d'exemple, l'interacteur *Gesture* produit un feedback de type trace et comprend un classificateur paramétrable qui interprète des séries d'événements souris en données textuelles ou en commandes. En outre, chaque interacteur connaît *plusieurs variantes* d'une même technique d'interaction, et peut être spécialisé par paramétrage. Voici, à titre d'exemple, les principaux paramètres partagés par tous les interacteurs :

- **Événements déclencheurs :** Ces paramètres définissent les événements d'entrée qui déclenchent, terminent et annulent l'interacteur : par exemple, les transitions d'un bouton de la souris ou du clavier.
- **Objets déclencheurs :** Ces paramètres déterminent au-dessus de quel(s) objet(s) graphique(s) doit se trouver le pointeur pour que l'interacteur démarre, et sur quel objet ce dernier opère : par exemple, un clic sur le fond d'une barre de défilement déplace le pouce d'un cran. Il permet également de définir des cibles illégales.
- **Discret/continu :** Ce paramètre spécifie si l'interacteur est uniquement activé au début et la fin de l'interaction, ou doit prendre en compte les déplacements du pointeur entre les deux.
- **Comportement en sortie :** Des paramètres permettent de spécifier ce qui se passe lorsque durant une interaction continue, le pointeur sort d'une région active. L'interaction peut par exemple continuer, être annulée (lorsque le pointeur sort d'un menu) ou la dernière valeur peut être conservée (lorsqu'il sort d'une barre de défilement).
- **Feedback :** Ces paramètres spécifient les objets à utiliser pour le feedback durant l'interaction ou à la fin de celle-ci. Par exemple, durant un cliquer-glisser l'objet-lui même est déplacé, ou

bien un objet fantôme est créé. À la fin d'une interaction de type sélection, l'élément sélectionné d'un menu peut être indiqué par un contour rectangulaire ou un rectangle d'inversion vidéo.

Les paramètres autorisent la plupart du temps des types de valeurs sophistiqués comme des groupes d'événements : par exemple, la valeur `:anykeyboard :except #\control-G` affectée au paramètre `:stop-event` terminera l'interaction lorsqu'une touche est appuyée sauf s'il s'agit de la combinaison *control+G*. Des *formules* (ou *contraintes*) peuvent également être spécifiées en tant que valeur[6]. À titre d'exemple, la technique d'interaction consistant à déplacer les objets du bouton gauche de la souris et à les redimensionner du bouton droit pourra être décrite ainsi [MYERS 90] :

```
(Create-instance 'move-or-grower Move-Grow-Interactor
 (:start-event '(:leftdown :rightdown)) ; Démarrer avec le bouton gauche ou droit.
 (:start-where '(:element-of all-objs-aggregate)) ; Démarrer sur tout objet du
                                                  ; conteneur graphique.
 (:grow-p (formula (eq :rightdown (gvl :start-char)))) ; Redimensionner si l'événement
                                                       ; initial est le bouton gauche,
                                                       ; sinon déplacer.
 (:window mywindow))
```

Les mécanismes d'aiguillage événementiel et les modes sont essentiellement gérés à travers le paramètre `Active`, qui associé à une formule permet l'activation ou la désactivation conditionnelle d'un interacteur, et à travers le paramètre `Priority`, qui permet de définir des niveaux de priorité lorsque plusieurs interacteurs sont candidats au même événement.

Tous les interacteurs sont écrits autour du même automate à états, reproduit sur la figure 2.24 page suivante. Cet automate suppose que toute interaction peut être *démarrée*, *arrêtée*, *terminée* ou *suspendue*. Implémenter un nouvel interacteur revient à associer une action à chaque transition dans cet automate à états.

De Lapidary à Gamut

Une myriade d'outils visuels de construction d'interface ont été développés comme compléments à *Garnet* et *Amulet*, et la plupart d'entre eux offrent une interface graphique aux interacteurs. Dans *Lapidary* [MYERS 90, ZANDEN & MYERS 95], l'outil de construction d'interfaces livré avec *Garnet*, les interacteurs peuvent être paramétrés à travers des boîtes de dialogue. Le modèle à contraintes de *Garnet* permet ainsi d'associer la programmation textuelle déclarative à la programmation visuelle.

Une partie des comportements, notamment certains types de feedback, peut être directement décrite graphiquement dans *Lapidary*. Par exemple, un feedback de type *case à cocher* sera dessinée et liée par des contraintes à un élément de menu présent à l'écran, et ces contraintes sont généralisées automatiquement à toute cible potentielle de l'interaction. Ce type de technique combinant manipulation graphique et inférence est connue sous le nom de *programmation par démonstration* [MYERS 92].

Les techniques de programmation par démonstration employées dans *Lapidary* seront reprises et étendues par la suite, dans *Tourmaline* [WERTH & MYERS 93], *Marquise* [MYERS *et al.* 93], *Pursuit* [MODUGNO 93],

[6] Les formules peuvent également effectuer des actions par des effets de bord, bien qu'*Amulet* fournisse également des abstractions permettant d'encapsuler des commandes dans des objets.

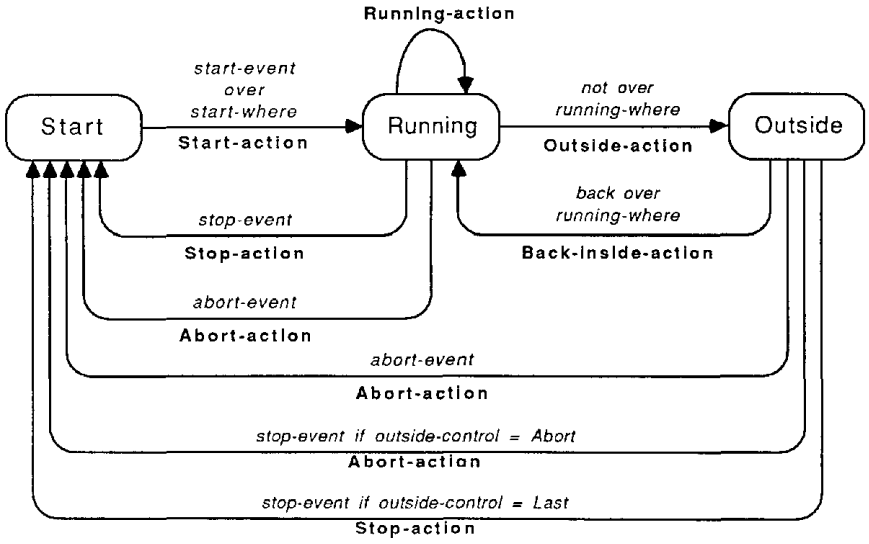


FIG. 2.24 – La machine à états décrivant le comportement générique d'un interacteur *Garnet*. Chaque transition est déclenchée par un événement, qui peut être dans certains cas spécifié par paramétrage, et déclenche une action, qui peut être décrite par programmation.

Silk [LANDAY & MYERS 95], *Topaz* [MYERS 98B], *Turquoise*, et enfin *Gamut* [MCDANIEL & MYERS 98]. L'objectif de ce dernier est de construire des jeux interactifs complets uniquement par démonstration, c'est-à-dire en donnant des exemples des comportements désirés. *Gamut* se distingue des autres outils par un moteur d'inférence extrêmement sophistiqué, qui repose sur des techniques d'intelligence artificielle. Ces techniques sont toutefois très dépendantes du domaine.

Discussion sur les apports et les limites de *Subarctic* et *Garnet/Amulet*

Ces deux boîtes à outils WIMP avancées procèdent d'approches différentes, mais chacune apporte des éléments essentiels dans la modélisation de l'interaction en entrée. L'approche de *Garnet* est intéressante car elle se veut la plus fidèle possible au modèle *MVC*. Dans l'implémentation originale de ce modèle en *SmallTalk* [KRASNER & POPE 88], la vue et le contrôleur étaient hautement dépendants l'un de l'autre, et le contrôleur devait être réimplémenté à chaque fois que la vue avait changé et vice-versa. Par la suite, la plupart des boîtes à outils s'inspirant de ce modèle, comme *Andrew* [PALAY *et al.* 88], *Interviews* [LINTON *et al.* 89] ou *Swing* [FOWLER 99] ont regroupé la vue et le contrôleur dans un même objet nommé *Vue*, *UI* ou *Look&Feel*. Dans *Garnet*, un protocole de communication assure une certaine indépendance entre la vue et le contrôleur, même si cette indépendance est restreinte par le fait que chaque type d'interacteur définit son propre protocole. Le grand mérite du modèle de *Subarctic* est quant-à-lui d'avoir explicité les mécanismes d'aiguillage standard qui, dans les boîtes à outils traditionnelles comme *Swing*, étaient implémentés de façon obscure, et par conséquent très difficiles à comprendre et à étendre.

Si *Subarctic* n'a pas pour objectif de rendre l'interaction personnalisable, le *paramétrage* est un aspect important de *Garnet*, et garantit une certaine configurabilité de l'interaction en entrée. Associé au paradigme de programmation par contraintes, il constitue de plus un outil extrêmement puissant. En

outre, il se prête relativement bien à un paradigme visuel de construction d'interfaces. Les paramètres des interacteurs de *Garnet* sont assez nombreux pour couvrir la plupart des techniques existantes. Cependant, lorsque le nombre de paramètres est excessif, le comportement d'un objet peut devenir difficile à comprendre et à configurer. C'est un peu le cas de certains interacteurs « à tout faire » de *Garnet*, comme le *Menu* ou le *Move-Grow*. Cela explique peut-être que les outils de visuels de *Garnet/Amulet* se soient orientés vers des techniques de programmation par démonstration.

Malheureusement, ces outils restent modérément ouverts à l'ensemble des paradigmes Post-WIMP. *Subarctic* prend en charge un ensemble très restreint de techniques post-WIMP et *Garnet* se limite à l'interaction gestuelle simple. Or la faiblesse majeure de l'approche de *Garnet*, en particulier, est le caractère très *ad-hoc* de son modèle d'entrée. Ses six types d'interacteurs se sont certes révélés suffisants dans les nombreux projets où ils ont été employés, mais ne sont pas adaptés à de nouveaux types d'entrée comme la parole [MYERS *et al.* 97], et n'encouragent pas la description de techniques d'interaction novatrices. Certains outils de base fournis pour le développement de nouveaux interacteurs sont également ad-hoc : par exemple, l'automate à états nécessite d'être étendu pour gérer des techniques comme le *rollover* ou l'*ajustement*[7].

Dans *Subarctic*, l'ajout de nouvelles techniques d'interaction, bien que facilité, n'est pas non plus gratuit. La plupart des nouveaux paradigmes d'interaction ne s'insèrent pas naturellement dans l'architecture existante, et nécessitent chacun une extension importante du modèle (*Multimodal Subarctic*, décrit dans la section 2.6.2 page suivante, en est une). Cela reste vrai pour des modifications mineures : par exemple, dans *Artkit* c'est la position initiale du geste qui détermine sa cible, et la prise en compte d'autres zones actives comme le centre du rectangle englobant nécessiterait l'ajout d'un nouveau type de politique d'aiguillage [TYSON R. *et al.* 90]. En outre, des problèmes de conflit qui interdisent par exemple l'abonnement simultané au clic et au double-clic nécessitent pour être résolus l'implémentation de nouveaux agents de médiation [TYSON R. *et al.* 90].

Mais l'aspect *dispositifs d'entrée* reste le facteur le plus limitant pour l'accès aux paradigmes Post-WIMP : ni *Subarctic* ni *Garnet* ne gèrent les dispositifs multiples et leurs modèles restent conçus pour les dispositifs standard. Peut-être par souci de portabilité, *Subarctic* repose exclusivement sur les mécanismes d'AWT et Swing, auxquels il délègue une partie de son modèle : ainsi, les événements natifs AWT sont passés en paramètre dans tous les événements *Subarctic*, afin notamment que les widgets puissent accéder aux touches modificatrices. En outre, le fait que les événements « bas-niveau » traités par *Subarctic* et *Garnet* soient en réalité des événements génériques standard (produits par Swing pour le premier et par X pour le second) ne permet pas d'exploiter au mieux les capacités des dispositifs même standard.

2.6.2 Les boîtes à outils Post-WIMP spécialisées

Contrairement aux boîtes à outils WIMP avancées, les boîtes à outils Post-WIMP sont conçues dès le départ pour décrire des paradigmes d'interaction non-conventionnels. Bien que certaines d'entre elles soient extensibles, elles reposent en général sur un paradigme ou un modèle d'interaction spécifique.

L'approche de *Multimodal Subarctic* [MANKOFF *et al.* 00], tout en étant relativement générale, intègre dans le mécanisme événementiel la notion d'*ambiguïté*, qui est propre à des modalités parti-

[7] Le *rollover* consiste à produire un effet graphique lorsque le pointeur entre dans le widget. L'*ajustement*, utilisé dans certaines barres de défilement, est un état où le modèle n'est pas mis à jour pendant l'interaction.

culières comme la parole ou le geste. Nous décrivons dans un premier temps cette boîte à outils qui constitue une contribution importante dans le domaine.

Les approches que nous décrivons par la suite sont plus spécifiques et concernent, dans l'ordre, l'*interaction gestuelle*, l'*interaction multi-pointeurs*, les *outils semi-transparents*, l'*interaction 3D* et les nouveaux paradigmes comme la *sensibilité au contexte* et les *interfaces tangibles*. Pour compléter, nous étudierons la question de l'*accessibilité*.

L'interaction ambiguë avec *Multimodal Subarctic*

Depuis *Artkit*, les efforts sur la boîte à outils *Subarctic* ont essentiellement porté sur l'aspect graphique [HUDSON & SMITH 96B]. Récemment cependant, *Subarctic* a été étendue pour prendre en compte de nouvelles modalités comme la parole et le geste [MANKOFF *et al.* 00]. Il s'agit d'une version non encore distribuée qui se distingue notablement de la version courante, et que nous baptiserons ici *Multimodal Subarctic*. Le modèle de gestion des entrées y a été entièrement remanié dans le but de gérer les *ambiguïtés* et les techniques de *médiation*, aspects essentiels de l'interaction multimodale. Nous exposons brièvement ses principes.

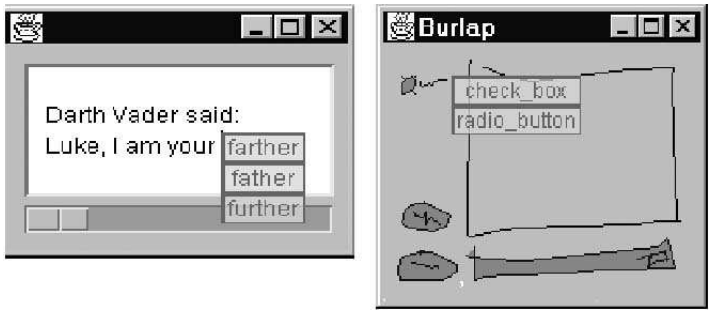


FIG. 2.25 – Une stratégie de médiation de type *n-best list* appliquée à une application de dictée vocale et une application de prototypage gestuel d'interfaces [MANKOFF *et al.* 00].

Avec des modalités telles que la parole ou le geste, des ambiguïtés peuvent survenir lors de la conversion d'un événement bas-niveau en un événement plus haut-niveau. Par exemple, une trace pourra être interprétable comme un cercle ou comme un rectangle, avec des indices de confiance comparables. Dans les applications existantes, l'ambiguïté est la plupart du temps résolue automatiquement en optant pour la probabilité la plus forte. Les techniques les plus efficaces consistent cependant à faire intervenir l'utilisateur, par exemple en lui proposant une liste de choix (figure 2.25), ou en lui demandant de répéter. Ce sont ces techniques, appelées *médiations*, que *Multimodal Subarctic* se propose de prendre en charge.

Dans la situation d'ambiguïté décrite précédemment, l'événement *cercle* et l'événement *rectangle* sont tous deux générés par *Multimodal Subarctic*, puis propagés et interprétés comme n'importe quel événement. Les événements ambigus étant provisoires, la réversibilité est assurée à travers un modèle à *événements hiérarchiques* [MYERS & KOSBIE 96], où un graphe orienté relie événements-source et événements interprétés. La résolution d'une ambiguïté (qui se fait plus tard, voir plus bas) consistera à *accepter* certains événements et en *rejeter* d'autres. L'acceptation d'un événement impliquera l'acceptation de ses événements-source, ainsi que le rejet des événements avec lesquels il est en conflit. Le rejet d'un événement conduira quant-à-lui au rejet de toutes ses interprétations.

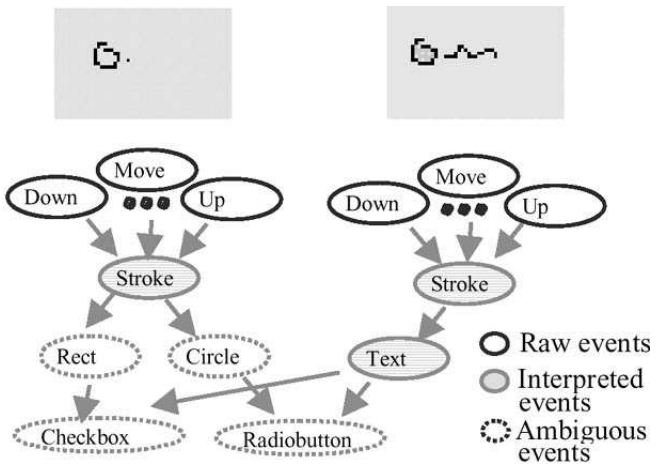


FIG. 2.26 – Événements hiérarchiques résultant de deux gestes à la souris [MANKOFF *et al.* 00].

La figure 2.26 reproduit une situation de prototypage gestuel d'interfaces, où une forme mi-rectangle mi-cercle est tracée par l'utilisateur, suivie d'un geste de type *texte* (en haut de la figure). Cette interaction peut signifier soit l'ajout d'une case à cocher, soit l'ajout d'un bouton radio. Les événements hiérarchiques créés sont représentés par un graphe dont les nœuds sont des événements de bas niveau (ellipses à contour noir), des événements synthétiques non ambigus (ellipses grisées) ou des événements synthétiques ambigus (ellipses à contour discontinu). Ici, l'acceptation de l'événement *Checkbox* induirait l'acceptation de l'événement *Rect*, puis le rejet de l'événement contradictoire *Circle*, impliquant à son tour le rejet de l'événement *Radiobutton*.

C'est le *système de médiation* qui se charge initialement d'accepter ou rejeter des événements. Il reçoit les événements ambigus après leur propagation, et les transmet successivement à ses *médiateurs*, des objets qui savent résoudre un type particulier d'ambiguïté. Un médiateur intéressé peut choisir de résoudre immédiatement l'ambiguïté (en général en interagissant avec l'utilisateur), ou bloquer cette ambiguïté pour la résoudre plus tard. Le système de médiation emploie de préférence des stratégies *paresseuses*, où la médiation est à la fois repoussée dans les couches d'abstraction supérieures et différée dans le temps, dans le but de :

- *Donner à l'ambiguïté l'occasion de se résoudre toute seule* : c'est le cas par exemple lorsqu'un seul événement ambigu est finalement consommé, les autres n'étant pas interprétables par les widgets.
- *Ne pas démarrer la médiation trop tôt* : par exemple, ne pas afficher un menu avant que le geste ne soit terminé. Ce dernier comportement est implémenté par le médiateur *StrokePauser* qui bloque les événements jusqu'à réception d'un événement *Release*.
- *Privilégier les médiations de haut-niveau* : par exemple, *« Est-ce une case à cocher ou un bouton radio ? »* est préférable à *« Est-ce un cercle ou un rectangle ? »*. De même, il est souhaitable de fournir des techniques de médiation dédiées à la tâche.
- *Produire un feedback complet* sur l'interprétation des différentes alternatives. Chaque widget peut ainsi effectuer un retour utilisateur afin de montrer ce qui se produirait si l'événement était accepté. Cela suppose des widgets spécialisés qui sachent traiter des événements provisoires, pour ensuite les accepter ou les rejeter.
- Enfin, *donner à l'utilisateur la possibilité de différer son choix*. Par exemple, une trace brute

possède un sens en tant qu'annotation, et n'a pas besoin d'être interprétée comme un bouton radio ou une case à cocher tant que l'utilisateur n'a pas décidé d'interagir avec.

L'interaction gestuelle

Le geste, cas particulier d'entrée ambiguë, est une modalité dominante dans les applications dites gestuelles et dans les systèmes à stylet. Si *Multimodal Subarctic* ainsi que *Subarctic* et *Amulet* proposent un ensemble minimal de techniques d'interaction gestuelle, certains outils plus conventionnels mais spécialisés offrent une prise en charge plus complète des diverses techniques gestuelles existantes. C'est le cas du système d'exploitation *PenPoint* [CARR 91] et des boîtes à outils *Flatland* [MYNATT *et al.* 99] et *Satin* [HONG & LANDAY 00]. Nous présentons ici *Satin*, dont l'architecture en entrée est assez représentative tout en étant l'une des plus abouties aujourd'hui.

Satin [HONG & LANDAY 00] est une librairie *Java* qui couvre la plupart des techniques gestuelles existantes (voir figure 2.27 page ci-contre) et les intègre en partie à *Swing*. *Satin* emploie son propre modèle graphique (avec prise en charge du dessin structuré, des couches, des vues multiples et des effets de zoom) et définit une architecture d'interaction gestuelle par-dessus les événements souris standard.

Les événements souris sont convertis en *traces* qui sont propagées dans la hiérarchie graphique de *Satin* comme des événements positionnels (à ceci près qu'une trace est uniquement transmise aux objets qui la contiennent entièrement) et traitées au sein de chaque objet par plusieurs types d'*interpréteurs*. Les *interpréteurs gestuels* comportent un *moteur de reconnaissance* qui à partir d'une trace produisent une liste de commandes reconnues, ordonnée par probabilité décroissante. Les *interpréteurs d'encre* comportent des algorithmes de traitement permettant de couper, joindre, simplifier des traces ou les transformer en segments.

Satin distingue également les *interpréteurs discrets* des *interpréteurs progressifs* qui peuvent effectuer des actions durant un geste, et introduit la notion de *multi-interpréteurs*, des interpréteurs composites munis d'une stratégie de choix. Cette dernière consiste en général à transmettre une trace aux interpréteurs-fils jusqu'à ce qu'elle soit consommée, mais peut également gérer des modes : le *multi-interpréteur zoom sémantique*, par exemple, active ou désactive des interpréteurs en fonction du niveau de zoom actuel.

Enfin, *Satin* ajoute à *Swing* un widget gestuel de type *Marking Menu*, et modifie dans son *« Pen Look & Feel »* certains widgets *Swing* dans le but de faciliter la manipulation au stylet (suppression du double-clic et élargissement de certains éléments).

L'interaction multi-pointeurs

Un petit nombre de prototypes boîtes à outils ont été proposées dans le but de décrire des techniques de travail collaboratif où plusieurs utilisateurs interagissent avec la même application sur le même poste de travail, ou pour décrire des techniques d'interaction bimanuelle. Il s'agit principalement de *MMM* [BIER & FREEMAN 91], *Bimanual Whizz* [CHATTY 94], et *MID* [HOURCADE & BEDERSON 99]. Ces trois outils ont en commun la prise en charge de dispositifs de pointage multiples.

MMM (Multi-Device Multi-User Multi-Editor). *MMM* [BIER & FREEMAN 91] est un prototype d'application interactive pouvant être contrôlée à la souris par plusieurs utilisateurs. Cette application

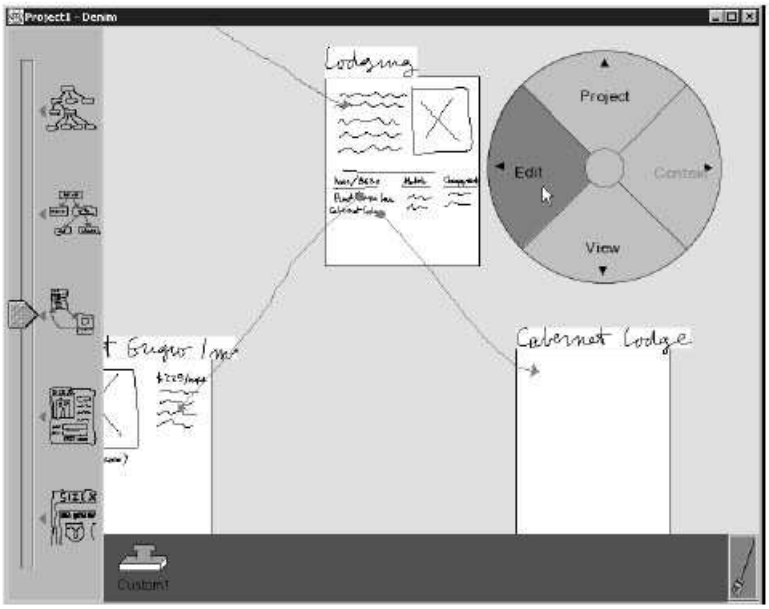


FIG. 2.27 – *Denim*, une application de prototypage de sites web basée sur les paradigmes d'interaction gestuelle et d'interface zoomable, développée avec la boîte à outils *Satin* [HONG & LANDAY 00].

consiste principalement en des menus et des hiérarchies d'*éditeurs*, qui sont des zones rectangulaires contenant des objets géométriques manipulables, des zones de textes ou encore d'autres éditeurs. Bien qu'il ne s'agisse pas à proprement parler d'une boîte à outils, *MMM* a été développé dans le but de valider à la fois un *modèle d'interaction* et une *architecture logicielle* adaptés à l'interaction multi-utilisateurs.

Le modèle d'interaction de *MMM* distingue tout d'abord utilisateurs et dispositifs : après s'être saisi d'une souris, chaque utilisateur s'enregistre en cliquant sur sa *zone personnelle*, un petit rectangle portant son nom (voir figure 2.28 page suivante). En outre, les *modes d'interaction sont dupliqués pour chaque utilisateur* : chaque utilisateur possède sa propre sélection (qui représente aussi le focus clavier et le focus commande pour les menus), ses attributs par défaut (police de texte, couleur), une position de caret (curseur textuel), et bien évidemment une position de pointeur. Chacun de ces modes comporte un feedback ; certains comme les attributs par défaut sont visualisés *dans la zone personnelle* de l'utilisateur (figure 2.28 page suivante), d'autres comme les pointeurs et les sélections sont affichés *dans la couleur personnelle* de l'utilisateur pour pouvoir être distingués. Ces derniers sont *superposables* graphiquement, y compris les sélections graphiques et textuelles. Les *menus* sont des objets partagés qui peuvent néanmoins être *dupliqués* par les utilisateurs pour être placés dans leur zone de travail. Enfin, *MMM* autorise l'interaction concurrente avec une granularité assez fine. Par exemple, un utilisateur peut déplacer un éditeur pendant que l'autre modifie l'un de ses objets.

Pour prendre en charge ces techniques, *MMM* repose sur un modèle de gestion des entrées spécifique. Les événements souris traditionnels ont été repris et étendus avec un champ *identifiant le dispositif*, un champ *identifiant l'utilisateur*, et un champ contenant *l'état des autres dispositifs*. Ces événements sont générés puis transmis à travers la hiérarchie des éditeurs jusqu'à être consommés, selon le mécanisme d'aiguillage classique. La différence est que chaque éditeur possède sa *propre*

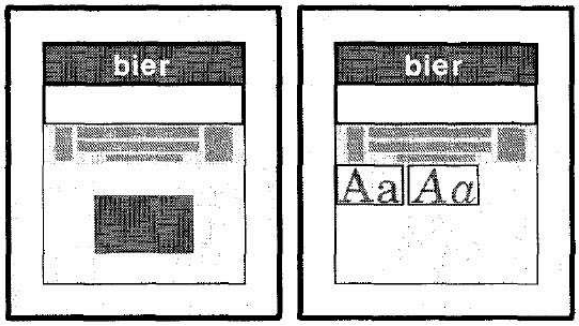


FIG. 2.28 – La *zone personnelle* d'un utilisateur dans *MMM*, à travers laquelle il peut s'enregistrer et y visualiser une partie de ses modes d'interaction. À gauche, un rectangle illustre la couleur de remplissage par défaut. À droite, la zone personnelle indique la police courante. [BIER & FREEMAN 91]

file d'événements qu'il interprète dans son *propre fil d'exécution*. À cette concurrence s'ajoutent des mécanismes de synchronisation qui permettent d'éviter les modifications incohérentes. D'abord, la *conversion en coordonnées locales* d'un événement positionnel lors de sa transmission à un éditeur n'est pas effectuée tant que la position de cet éditeur risque d'être *en cours de modification*, c'est-à-dire tant que le processus de son parent est actif (les événements clavier par contre, sont transmis sans attendre). Ensuite, l'événement est interprété en deux temps par l'éditeur : lorsqu'une modification de la sélection ou du focus positionnel est nécessaire, l'éditeur envoie une requête de mise à jour à un processus extérieur avant de replacer l'événement dans sa file pour une seconde phase de traitement. Enfin, le réaffichage est également effectué de manière asynchrone, selon un mécanisme analogue à celui de Java Swing [ECKSTEIN & LOY 02].

Bimanual Whizz. *Bimanual Whizz* [CHATTY 94] est le nom que nous donnons à l'extension bimanuelle de *Whizz* [ESTEBAN 97]. *Whizz* est une boîte à outils qui décrit l'interaction par un modèle à flot de données, et est dotée d'un outil de programmation visuelle (voir la section 2.7.2 page 83 sur l'éditeur graphique *Whizz'Ed*). *Whizz* repose essentiellement la boîte à outils X_{TV} [BEAUDOUIN-LAFON *et al.* 90], écrite pour le développement d'applications à manipulations directe avec X. X_{TV} constitue une bonne infrastructure pour la gestion de dispositifs multiples car ses événements possèdent la notion de *dispositifs-source*.

Bimanual Whizz implémente un modèle d'interaction bimanuelle qui distingue trois paradigmes [CHATTY 94] : l'interaction *indépendante* (les deux mains sont employées en série, par exemple pour sélectionner un outil avant de l'utiliser), l'interaction *parallèle* (les deux mains effectuent simultanément des tâches distinctes) et l'interaction *combinée* (les deux mains collaborent sur la même tâche). Son modèle d'interaction prend également en compte l'aspect asymétrique de l'interaction bimanuelle, en remplaçant pour la main non-dominante le *point actif* du pointeur par une *zone circulaire*.

Bimanual Whizz modifie peu le modèle à événements standard de X_{TV}, d'abord pour préserver la compatibilité avec le modèle original, et ensuite parce que ce modèle possède déjà une notion de dispositifs qui permet de distinguer des événements souris de sources différentes. Dès lors, il est possible de résoudre certains problèmes de concurrence, et de décrire par exemple des boutons sensibles à n'importe quel événement *Press* mais filtrant ensuite tout événement autre que le *Release* du dispositif à l'origine du *Press*. Les interactions parallèles continues de type cliquer-glisser posent d'autres problèmes de concurrence qui sont résolus dans *Bimanual Whizz* en instanciant à chaque interaction

un objet analogue à un interacteur de *Garnet*, qui prend en charge de façon autonome les transitions d'états, la manipulation et le feedback.

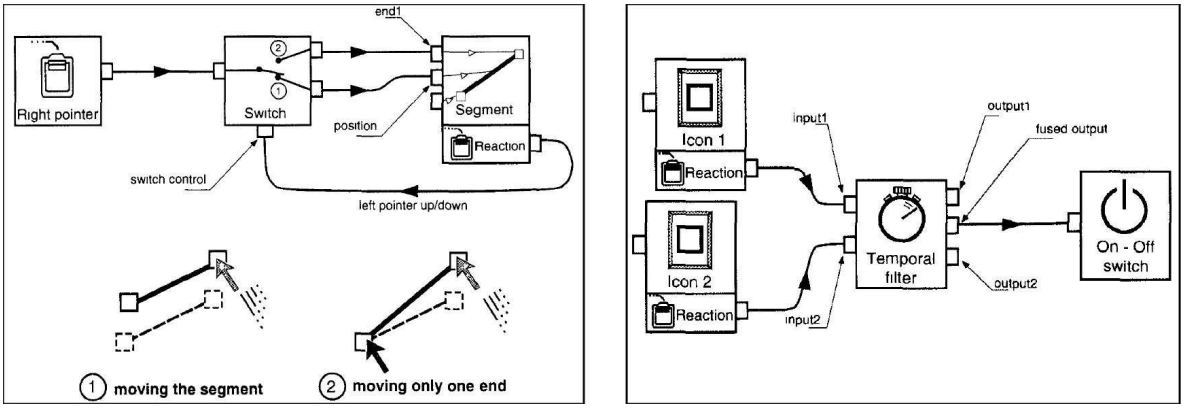


FIG. 2.29 – Utilisation de filtres dans *Bimanual Whizz* pour décrire des techniques d'interaction bimanuelle combinée. Sur le *schéma de gauche*, le pointeur droit est utilisé pour déplacer un segment ; lorsque le pointeur gauche clique sur une extrémité de ce segment, le flot est redirigé par le filtre *Switch* et le segment est déformé. Le *schéma de droite* décrit une technique de fusion temporelle, où un filtre spécialisé ferme l'application lorsqu'il reçoit deux événements dans un intervalle de temps spécifié. [CHATTY 94]

Pour les *interactions combinées*, *Bimanual Whizz* distingue deux types de fusion. Dans une combinaison de type *mode+événement*, le premier dispositif spécifie un mode d'interaction pour le second : par exemple, la manipulation de l'extrémité d'un segment est interprétée comme une translation sauf si l'autre extrémité est maintenue par un autre pointeur. Le modèle de *Whizz* permet d'introduire aisément ce type de mode par l'ajout d'un module dans le flot de données (voir figure 2.29, image de gauche). Dans une combinaison de type *événement+événement*, deux événements simultanés sont interprétés en un événement de plus haut niveau : par exemple, cliquer simultanément sur deux boutons peut fermer l'application. Ce type de fusion temporelle nécessite simplement dans *Whizz* la définition d'événements synthétiques supplémentaires et l'ajout d'un filtre dans le flot de données (voir figure 2.29, image de droite).

MID (Multiple Input Devices). *MID* [HOURCADE & BEDERSON 99] est une librairie qui étend le modèle *AWT* de Java pour gérer des dispositifs de pointage multiples. Elle ne définit pas de nouveaux types d'événements, mais étend la définition d'un événement souris en y introduisant la notion de *dispositif-source*, selon une approche très similaire à *Bimanual Whizz*. Les applications Java existantes nécessitent peu de modifications pour être rendues compatibles avec *MID*, et seules les opérations consistant, pour les widgets, à s'enregistrer comme observateurs d'événements souris nécessitent d'être traduites. L'application peut ensuite être modifiée afin d'exploiter les identifiants de dispositifs. *MID* a servi à implémenter une application de dessin multi-utilisateurs nommée *KidPad*.

MID introduit dans l'*AWT* la notion de dispositifs de pointages multiples d'une manière relativement séduisante, sans modifier les principes fondamentaux de cette boîte à outils. Cependant, l'approche de *MID* est essentiellement pragmatique et de bas-niveau. En particulier, il n'aborde pas les problèmes de la gestion concurrente des événements positionnels que nous avons précédemment évoqués.

Les outils semi-transparents

Les boîtes à outils que nous décrivons ici constituent en quelque sorte des évolutions des boîtes à outils multi-pointeurs : elles gèrent l'interaction bimanuelle, mais y ajoutent de nouvelles techniques d'interaction basées sur la semi-transparence (voir section 1.3.3 page 19). Elles sont cependant peu représentées. En introduisant le premier modèle d'interaction basé sur ce paradigme, Bier et al [BIER *et al.* 93] décrivent par la même occasion une *extension de la boîte à outils MMM* utilisée pour implémenter ces techniques. Nous la décrivons dans un premier temps. Par la suite, Kurtenbach et al. [KURTENBACH *et al.* 97] ont étendu de façon intéressante ce modèle d'interaction bimanuelle avec leur application *T3* (Tablets, Two-hands, Transparency), notamment en augmentant le nombre de degrés de contrôle (deux dispositifs positionnels sensibles à la rotation sont employés). Cependant, ils ne proposent pas de nouvelle boîte à outils. Enfin, une autre application-prototype, *CPN2000* [BEAUDOUIN-LAFON & LASSEN 00], a été développée sur un modèle d'interaction similaire, et a permis d'introduire une nouvelle architecture de boîte à outils Post-WIMP. Nous décrivons également cette architecture.

MMM remanié. Le modèle d'interaction bimanuelle basé sur les *outils semi-transparents* a été décrit par Bier et al [BIER *et al.* 93] dans le but de montrer comment l'emploi combiné de palettes flottantes semi-transparentes composées d'outils divers et de lentilles magiques peut faciliter des tâches de création, d'édition, ou de sélection d'objets graphiques. Ces interactions ont été implémentées avec la boîte à outils *MMM* (voir section 2.6.2 page 70), après des remaniements importants de son modèle d'affichage et de son modèle d'entrée.

Cette boîte à outils a été tout d'abord modifiée pour interpréter de façon spécifique les événements en provenance d'un dispositif *trackball*, afin qu'il puisse être employé par la main non-dominante pour déplacer et redimensionner les objets flottants et naviguer dans la fenêtre graphique. En outre, l'introduction d'outils flottants déstructure le modèle hiérachique des objets graphiques pour lequel le mécanisme d'aiguillage était conçu (voir figure 2.30). Les événements positionnels doivent ainsi *remonter* vers l'application après avoir été transmis aux objets graphiques et éventuellement modifiés par ceux-ci. Les deux types de modification sont l'*annotation* et le *changement de position*.

L'*annotation* a pour but de s'assurer qu'un objet ne recevra pas deux fois le même événement, et permet de transmettre des commandes. Une palette flottante, par exemple, ajoute une commande à l'événement et le retourne à l'application, qui se charge ensuite de passer cet événement aux objets qui se trouvent en-dessous et ainsi de suite. Les commandes sont concaténées pour permettre l'usage combiné de plusieurs palettes. Quant aux lentilles magiques, dont la plupart sont déformantes, elles modifient la position de chaque événement afin qu'il soit correctement dirigé vers l'objet qui apparaît sous sa position.

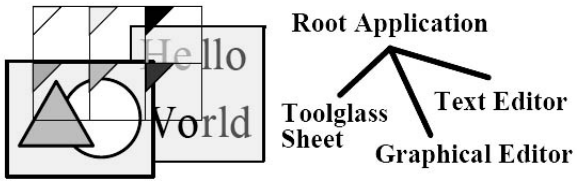


FIG. 2.30 – Les outils flottants ajoutent à la notion de hiérarchie celle de *couche graphique* : les trois outils représentés ici sont au même niveau dans la hiérarchie des widgets, bien qu'ils soient superposés et affichés dans un ordre déterminé (de droite à gauche dans l'arbre) [BIER *et al.* 93].

L'architecture CPN2000. *CPN2000* [BEAUDOUIN-LAFON & LASSEN 00] est une application complète d'édition de réseaux de Pétri colorés, employant un modèle d'*interaction instrumentale* (voir section 2.5.2 page 57) mêlant efficacement manipulation directe et bimanuelle, palettes d'outils conventionnelles ou semi-transparentes, *Marking Menus* (voir section 1.3.2 page 18), guides magnétiques de type *Snap-Dragging*, et un modèle de documents basé sur une métaphore de classeurs. Ce modèle privilégie la redondance (il existe de multiples manières d'effectuer une tâche) et autorise par des quasi-modes l'accès simultané à plusieurs outils.

L'architecture de *CPN2000* est de type *MVC*, avec une structure de document (M), une structure d'affichage (V) et une structure d'entrée (C) clairement séparés et faiblement couplés. La structure d'entrée est composée d'*instruments d'interaction*, qui gèrent des entrés physiques et possèdent une présentation. Les instruments consultent la structure graphique pour le *picking* mais opèrent directement sur les éléments (*nœuds*) du document structuré avec un protocole simple à base de *commandes*.

Plusieurs instruments spécifiques à une classe particulière de nœud peuvent être combinés en un *instrument générique*. En outre, un instrument est lui-même un document, ce qui lui permet d'être manipulé par d'autres instruments (par exemple, les outils d'une palette peuvent être déplacés avec un instrument *Move*). Pour finir, l'activation et la désactivation de chaque instrument est géré par deux machines à états, un par dispositif.

L'interaction 3D

Par opposition aux scènes 3D statiques, les scènes animées en temps réel comportent des éléments (caméras, sources lumineuses, objets) dont les attributs (position, forme, couleur) sont variables au cours du temps. Dans le domaine de la 3D, l'interaction est considérée comme un cas particulier d'animation, où les attributs n'évoluent plus de façon autonome mais en fonction de valeurs en provenance des dispositifs d'entrée ou de dispositifs « virtuels » (la plupart du temps, des widgets 2D). Ces dispositifs sont vus comme des groupes de canaux produisant des valeurs de façon continue.

Ainsi, la majorité des nombreuses boîtes à outils 3D modernes décrivent l'interaction par des *contraintes* ou des *flots de données* liant des canaux à des attributs : citons par exemple *UGA* [ZELEZNIK *et al.* 91], *Inventor* [STRAUSS & CAREY 92], *TBAG* [ELLIOTT *et al.* 94], *WorldToolkit* [SENSE8 03], *Virtools* [VIRTOOLS 01] ou des approches plus récentes comme *OpenTracker* [REITMAYR & SCHMALSTIEG 01] et *3dml/inTml* [FIGUEROA *et al.* 02]. Un petit nombre d'opérateurs mathématiques simples (par exemple, des opérateurs permettant d'agir sur la vitesse ou l'accélération d'un objet plutôt que sur sa position) suffit à décrire la plupart des techniques de *contrôle continu et statique* tels qu'on les trouve dans les outils de navigation 3D ou les jeux. L'essentiel du comportement de ce type d'application est pris en charge par des outils de simulation mécanique, et les actions discrètes sont traitées comme de simples commandes (par exemple, l'appui d'un bouton lance l'animation d'un tir de missile).

En revanche, les applications de conception et d'édition où des objets sont manipulés ont des besoins plus proches de ceux des interfaces 2D, et les *modes* y jouent notamment un rôle important. Les modes et en particulier le multiplexage temporel et spatial sont habituellement traités dans les boîtes à outils 3D comme des remaniements explicites du graphe de contraintes ou des reconnexions dynamiques dans le flot de données. Les stratégies de picking employées pour les dispositifs 2D sont similaires à celles des boîtes à outils conventionnelles, alors que la sélection 3D nécessite des techniques plus avancées comme le *Go-Go*[8], techniques qui sont rarement prises en charge.

[8] Le Go-Go est une technique de sélection 3D où le bras virtuel est étendu jusqu'à atteindre l'objet [POUPYREV *et al.* 96].

D'autres paradigmes en cours d'exploration

De nouvelles boîtes à outils continuent à être proposées pour prendre en charge, du moins en partie, de nouveaux paradigmes d'interaction encore en cours d'exploration. Nous décrivons brièvement le *Context Toolkit* utilisé pour construire des applications *sensibles au contexte* et les *Phidgets*, employés pour construire des *interfaces tangibles*.

Context Toolkit. Les applications *sensibles au contexte* (voir section 1.3.5 page 24) sont difficiles à construire, étant donné leur nature *distribuée* et l'emploi de dispositifs non-standard (capteurs) de nature très *hétérogène*. Le *Context Toolkit* [SALBER *et al.* 99] propose une architecture basée sur la notion de *widgets de contexte*, qui communiquent entre eux par *TCP/IP* et *XML*.

Les widgets de contexte maintiennent un état en fonction de données en provenance de *générateurs*, qui sont des abstractions des senseurs physiques. Ils se chargent également de notifier l'application de tout changement dans cet état. Par exemple, le widget *Présence* notifie de l'identité de chaque arrivant et son heure d'arrivée dans un lieu donné à partir de systèmes à badge ou d'analyse biométrique, et le widget *Activité* informe de tout changement dans le niveau d'activité en fonction de données en provenance de microphones ou de caméras vidéo.

Les widgets de contexte sont composables : par exemple, un widget *Réunion* peut détecter le début ou la fin d'une réunion en fonction du nombre de personnes dans la salle et du niveau d'activité. Les *Serveurs* sont des conteneurs plus importants de widgets, qui peuvent par exemple notifier l'application de chaque début et fin de réunion dans l'ensemble d'un immeuble.

Les Phidgets. Dans les *interfaces tangibles* (voir section 1.3.5 page 23), une même métaphore peut être mise en œuvre par des techniques très diverses. Les interfaces basées sur la capture vidéo nécessitent avant tout des librairies qui prennent en charge l'acquisition, l'analyse et l'interprétation d'images en provenance de caméras ou web-cams. D'autres interfaces font usage de dispositifs spécialisés comme les capteurs 3D, d'autres encore reposent sur des dispositifs « maison ». Si les dispositifs exotiques du commerce sont souvent ardus à programmer, il est encore plus difficile de construire soi-même des dispositifs même simples.

L'objectif des *Phidgets* [GREENBERG & FITCHETT 01] est de fournir des composants physiques agençables prêts à l'emploi et aisément accessibles à travers une API unifiée. La partie physique des phidgets communique à travers le protocole *USB*, et la partie logicielle repose sur les standards *COM* et *ActiveX* de *Microsoft*. Le composant logiciel principal, le *gestionnaire de phidgets*, notifie l'application des connexions et déconnexions de phidgets et instancie un *objet d'accès* à chaque connexion. Ce dernier comporte une interface générique permettant notamment d'identifier le dispositif de façon unique, et une interface spécifique permettant de lire et/ou modifier son état, et être notifié de ses changements.

Un phidget peut également être relié et synchronisé à un *widget* : Par exemple, un *slider* peut être utilisé pour contrôler un potentiomètre motorisé, visualiser sa position, ou simuler son comportement lorsqu'il est absent. La librairie *WidgetTap* [GREENBERG & BOYLE 02] offre maintenant des outils d'inspection permettant d'attacher des phidgets à des widgets d'applications *Windows* conventionnelles (voir également la section 1.3.5 page 23 à ce sujet).

L'accessibilité

S'il n'existe pas à notre connaissance de boîte à outils prenant spécifiquement en charge des techniques d'interaction adaptées à des entrées *appauvries*, certaines boîtes à outils proposent des solutions simples pour améliorer l'accessibilité. La plupart d'entre elles offrent ainsi aux utilisateurs experts et ceux ne pouvant employer une souris des moyens d'interagir avec les widgets par l'intermédiaire du clavier.

L'*API d'accessibilité* de *Swing* [SUN 02] va plus loin en définissant un contrat entre l'interface et les technologies d'accessibilité extérieures, qui consiste principalement à attribuer à chaque widget un nom, une description et des moyens de naviguer dans sa hiérarchie. L'objectif de cette API est de fournir une « vue » textuelle des applications pour qu'elles puissent être perçues et contrôlées indépendamment de leur présentation graphique.

Aucune de ces techniques n'étant en mesure de garantir l'accessibilité effective d'une application, les *guides d'accessibilité* publiés à l'usage des développeurs en constituent un complément indispensable [DUNN 00, SCHWERDTFEGER 00]. Ces derniers conseillent par exemple de s'assurer que tout widget peut être traversé par le mécanisme de focus, et que des raccourcis sont définis pour tout élément de menu et fonction importante de l'application.

Discussion sur les apports et les limites des boîtes à outils spécialisées

Malgré leur spécialisation, les boîtes à outils Post-WIMP contribuent de façon importante à la compréhension de l'interaction en entrée. L'apport de *Multimodal Subarctic*, par exemple, est d'avoir rationalisé les techniques de *médiation* qui étaient déjà employées par certaines applications mais mal comprises. Cette boîte à outils introduit une architecture claire et propose des outils efficaces pour la gestion des *entrées ambiguës*, et est par conséquent bien adaptée au paradigme de l'*interaction multimodale*. Un autre apport de ce modèle est d'avoir explicité un aspect à notre avis important de l'interaction en entrée, à savoir l'existence de *niveaux d'abstraction* définis par des *chaînes de traitements successifs*. Cette notion n'apparaissait que de façon implicite dans *Subarctic* où chaque agent recevait les mêmes événements bas-niveau. Ces chaînes de traitement deviennent réellement explicites dans le modèle à flot de données de la boîte à outils *Whizz* et son extension à l'interaction bimanuelle. L'architecture de *CPN2000* offre quant-à-elle une base solide aux techniques de type instrumental et décrit une implémentation efficace du modèle MVC.

Les boîtes à outils spécialisées sont plus ouvertes aux entrées que les boîtes à outils WIMP, mais chacune d'entre elles comporte ses limites. Bien que *Multimodal Subarctic* ait ajouté des modalités (parole, geste et contexte) à *Subarctic*, sa prise en charge des dispositifs non-standard reste limitée, et le clavier et la souris continuent à y jouer un rôle prépondérant. Comme dans *Satin*, l'interaction gestuelle repose exclusivement sur les événements souris conventionnels. Les boîtes à outils multi-pointeurs et de semi-transparence exploitent efficacement des entrées positionnelles multiples, mais de façon ad-hoc et avec des dispositifs de type spécifique : *MMM* décrit des techniques adaptées à l'interaction multi-utilisateurs avec la souris et sa version bimanuelle gère un *trackball* et une souris de manière câblée. *Bimanual Whizz* ne reconnaît que les souris, bien qu'il les gère de façon très extensible.

Les boîtes à outils 3D sont celles qui savent le mieux exploiter les entrées enrichies. Leurs modèles à canaux permettent d'obtenir, avec peu d'abstractions, une certaine indépendance par rapport aux

entrées, que certains outils comme *WorldToolkit* exploitent au maximum en prenant en charge un grand nombre de dispositifs. Ces modèles autorisent en outre une grande flexibilité dans la description d'interactions et se prêtent bien à la programmation visuelle (voir plus loin, section 2.7 page ci-contre). Cependant, leur pouvoir d'expression se limite au contrôle d'attributs d'objets 3D, et ces outils ne sont pas adaptés à la complexité de l'interaction 2D. Bien qu'il prenne en charge moins de dispositifs non-standard, *Bimanual Whizz* tire avantage du même type de modèle à flot de données pour décrire certaines techniques d'interaction 2D Post-WIMP.

Aucun de ces outils, y compris les plus ouverts en entrée, n'a été prévu pour l'accessibilité. Les recherches actives dans ce domaine se posent généralement pour objectif de définir des architectures applicatives où l'on puisse greffer des modalités de sortie et d'entrée alternatives, ce qui suppose tout d'abord de rendre le modèle de l'application (ou la structure du document, pour les navigateurs Web) explicite. Si les implémentations incomplètes du modèle MVC comme celui de *Swing* autorisent dans une certaine mesure la substitution des présentations (utiliser une modalité sonore au lieu de visuelle, par exemple), elles n'externalisent pas suffisament le contrôle pour être exploitables du point de vue des entrées.

L'introduction de l'*API d'accessibilité* a néanmoins fait de Swing une boîte à outils considérée comme l'une des plus avancées dans le domaine, car l'on considère actuellement qu'une application est accessible dès lors qu'elle est compatible avec les technologies d'accessibilité existantes comme les « lecteurs d'écran ». Malgré tout, une réelle accessibilité exige des applications qu'elles proposent des techniques d'interaction adaptées à la fois au handicap et à la tâche. Or l'accessibilité des applications repose toujours sur des outils extérieurs, et nous ne connaissons à ce jour aucune boîte à outils capable d'aider le développeur à construire des applications directement accessibles.

2.6.3 Conclusion

Alors que chaque boîte à outils tente de proposer un modèle graphique le plus générique possible, chacune d'entre elles emploie un modèle d'entrée *ad-hoc*, qui convient à *« ce que l'on veut faire »*. Certaines boîtes à outils tentent certes de proposer des mécanismes généraux, mais sans réellement y parvenir. Ainsi, si les techniques gestuelles simples se décrivent bien dans les boîtes à outils WIMP avancées, la plupart des autres paradigmes nécessitent des changements incrémentaux importants. Le fait que nombreuses de ces extensions fassent l'objet de publications complètes est assez révélateur.

Cette grande hétérogénéité dans les boîtes à outils est naturelle. Tout d'abord, les paradigmes Post-WIMP sont très variés et leur unification pose un problème évident de complexité. Même si des applications comme *CPN2000* parviennent à composer des techniques multiples, certaines publications font état de la complexité de combiner deux paradigmes comme l'interaction gestuelle et coopérative [HONG & LANDAY 00]. En outre, un développeur d'outils se fixe en général pour objectif soit de prendre en charge les techniques propres à un paradigme d'interaction dans lequel il est expert (interfaces conventionnelles pour *Garnet*, interfaces gestuelles pour *Satin*), soit de prendre en charge un nouveau modèle d'interaction à des fins de démonstration (*MMM remanié*).

Les boîtes à outils Post-WIMP n'en permettent pas moins de construire des applications bien plus contrôlables dans le sens où elle prennent en charge des techniques d'interaction avancées. Cette contrôlabilité est néanmoins limitée par le fait que ces techniques s'appuient soit sur des dispositifs conventionnels (boîtes à outils gestuelles), soit sur un ensemble fixe de dispositifs qu'elles exploitent de façon efficace mais rigide (boîtes à outils multi-pointeurs). La version étendue de *MMM*,

par exemple, gère de façon très spécifique le trackball d'une part, et les pointeurs conventionnels d'autre part.

Seules les boîtes à outils 3D parviennent à obtenir une certaine indépendance par rapport aux entrées en permettant de connecter librement des dimensions physiques multiples à des attributs d'objets 3D. Et bien qu'elles soient loin de balayer l'espace des dispositifs non conventionnels existants, certaines d'entre elles prennent en charge des dispositifs 3D nombreux et divers. Elles exploitent cependant un paradigme Post-WIMP spécifique (le contrôle parallèle et direct) en ignorant presque tout des techniques avancées du domaine de la 2D (outils transparents, gestes, reconnaissance vocale, etc).

Aucune boîte à outils WIMP avancée ou Post-WIMP ne prend en charge l'accessibilité, c'est-à-dire l'interaction avec des entrées appauvries, autre que par des moyens standard (raccourcis clavier). La plupart offrent une certaine configurabilité orientée-programmeur dans la mesure où elles sont un peu plus extensibles et mieux construites que les boîtes à outils traditionnelles. Cependant, la grande majorité ne permet pas de configurer l'interaction sans programmation. Les exceptions sont *Garnet/Amulet*, qui permet de paramétrer les interacteurs, et les outils 3D qui offrent parfois un éditeur interactif, approche dont s'inspire également *Whizz'Ed* dans la 2D. Les éditeurs visuels d'interaction connus seront décrits de façon plus générale et détaillée dans la section suivante.

2.7 Les éditeurs graphiques d'interaction

Les éditeurs graphiques d'interaction permettent de construire ou de modifier des applications interactives ou des parties d'applications interactives par manipulation, assemblage, et connexion de blocs élémentaires.

Les outils de construction d'interfaces de type *Visual Basic* [FRANTZ 00] ou *Visual C++* [CHAPMAN 03] ne permettent de décrire que la partie *présentation* des interfaces graphiques, et ne seront pas évoqués ici. Parmi les outils permettant de spécifier le *comportement* des interfaces, nous distinguons deux catégories : les *éditeurs de simulations interactives 2D* qui permettent de spécifier simultanément la présentation et le comportement d'applications interactives 2D, et les *éditeurs de comportement 3D* qui sont destinés à la description de comportements d'applications 3D.

Chaque éditeur possède un style de programmation visuelle qui lui est propre, selon qu'il met l'accent sur l'aspect flot de données, flot de contrôle ou encore la programmation par contraintes. Certaines de ces approches sont communes aux éditeurs 2D et 3D. Nous les décrivons dans un premier temps. Puis, nous présentons les deux catégories d'éditeurs, en décrivant les principaux outils existants puis en analysant leurs avantages et leurs inconvénients.

2.7.1 Les paradigmes de programmation visuelle

Les éditeurs graphiques de comportements emploient des objets graphiques interconnectés assimilables à des graphes. La sémantique de ces graphes diffère selon l'approche de programmation visuelle utilisée :

Flot de données. Dans l'approche à flot de données, les sommets représentent des opérations atomiques et les arcs représentent des transferts de données entre ces opérations. Les sommets produisent de façon répétitive des valeurs en sortie en fonction des valeurs reçues en entrée.

Des *règles de déclenchement* spécifient à quel moment un sommet est activé. Le paradigme de flot de données permet de mettre en évidence les flux d'informations dans un processus, et décrit de façon explicite la dépendance entre les données. Il se focalise sur la *transformation*.

Programmation fonctionnelle. L'approche fonctionnelle est très proche de l'approche à flots de données, dans le sens où elle décrit également des transformations. Cependant, dans les graphes fonctionnels, les données sont « tirées » au lieu d'être « poussées » : une opération n'est activée que lorsque ses résultats sont nécessaires à une opération ultérieure. Ce mécanisme est également appelé évaluation paresseuse.

Flot de contrôle. Dans l'approche à flot de contrôle de type *organigramme*, les sommets décrivent également des opérations. Cependant, l'accent est mis sur le séquencement de ces opérations, et les flux de données disparaissent. Les arcs spécifient des relations d'ordre du type "s'exécute avant" et les données consommées et produites par les opérations sont reléguées dans des variables globales. Dans la version *automate* des graphes à flot de contrôle, les sommets décrivent des *états* et les arcs des *transitions* entre états pendant lesquelles sont effectuées les opérations (voir section 2.3.1 page 44). Ces opérations manipulent également des variables globales. Les deux types de graphes imposent un ordre total sur les opérations : un seul sommet est activé à la fois. Ce paradigme met en évidence le séquencement des opérations dans un processus. Il se focalise sur l'aspect *procédural*.

Programmation par contraintes. Les *contraintes* sont des relations portant sur une ou plusieurs variables, qui sont spécifiées indépendamment de toute notion algorithmique. Un *solveur* se charge de les *résoudre* (i.e. trouver une instanciation correcte des variables), ou de les *maintenir* (i.e. instancier en continu un sous-ensemble des variables en fonction d'autres variables qui évoluent au cours du temps). Dans les langages graphiques inspirés de la PPC, les sommets sont des variables et les arcs des contraintes qui devront être maintenues entre ces variables. Ces graphes n'imposent pas de relation d'ordre du type "s'exécute avant". La programmation par contraintes met en avant une description déclarative des problèmes, en termes de *relations* entre les composants.

2.7.2 Les éditeurs de simulations interactives

Les éditeurs de simulations interactives emploient des langages visuels où certaines primitives graphiques sont manipulables à l'exécution, ce qui permet de décrire simultanément la *présentation* et le *comportement* d'interfaces graphiques. Ils peuvent être employés pour décrire le comportement de widgets mais également la façon dont ils sont contrôlés par des événements de bas-niveau.

La plupart de ces éditeurs emploient des paradigmes de programmation par contraintes, d'autres ont une approche flot de données, d'autres encore ont une approche mixte. Dans cette section, nous décrivons ces trois types d'éditeurs.

Les contraintes de ThingLab

Thinglab [BORNING 79] est l'un des premiers systèmes à employer des techniques de programmation visuelle pour décrire des comportements interactifs. Implémenté dans le langage SmallTalk, Thinglab repose sur la PPC tout en restant volontairement très proche des concepts de la programmation orientée objet : classes, objets, méthodes, hiérarchie d'héritage et hiérarchie compositionnelle.

Les objets prédéfinis de ThingLab, pour la plupart des objets géométriques simples, consistent en des structures d'objets liés par des contraintes, et possédant une représentation graphique manipulable. De nouvelles classes d'objets peuvent être construites par composition, en assemblant graphiquement plusieurs objets existants et en spécifiant des contraintes supplémentaires.

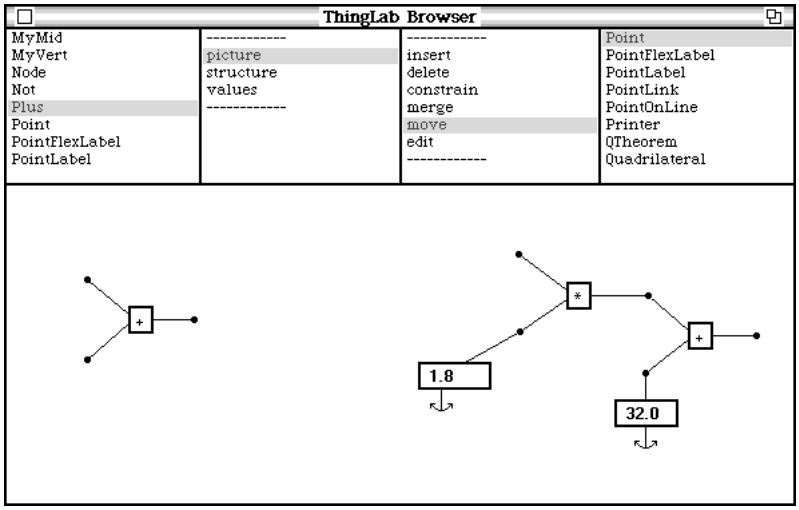


FIG. 2.31 – Une fenêtre de l'éditeur de simulation *Thinglab*. [BORNING 79]

Une fenêtre de ThingLab est représentée sur la figure 2.31. La partie supérieure de la fenêtre contient l'explorateur d'objets. Les classes de ThingLab sont listées dans le premier panneau, à gauche. Le second panneau permet d'afficher l'objet courant sous différentes formes (graphique, structurelle ou valeurs), le troisième permet d'appeler ses méthodes, et le dernier contient les arguments de ces méthodes. Une partie importante de l'interaction (instanciation et composition d'objets, définition des contraintes) s'effectue par appel de ces méthodes. La partie inférieure de la fenêtre contient la représentation graphique manipulable de l'objet courant.

À gauche de la zone inférieure, un opérateur d'addition a été construit par composition d'objets élémentaires. Ces objets élémentaires sont le point numérique (point géométrique possédant une valeur numérique interne) et la piste numérique (association d'un point numérique et d'un segment), ainsi qu'un cadre permettant d'afficher un symbole. L'opérateur est construit en composant trois pistes numériques et un cadre, et en spécifiant les contraintes géométriques et numériques qui les lient. Une piste numérique peut être affichée et rendue éditable en lui associant un champ de saisie. À droite, un convertisseur de températures a été construit par assemblage de ces objets. Les zones de saisie, fixées par des ancres, sont considérées comme des constantes par le solveur.

La figure 2.32 page suivante illustre deux autres exemples construits avec ThingLab. L'exemple de gauche est un document comportant quatre nombres visualisés sous deux formes : une forme textuelle et une forme graphique. Ces deux vues sont manipulables et synchronisées. L'exemple de droite est une fenêtre découpée en zones redimensionnables, construit en composant puis en contraignant des rectangles.

Thinglab a eu un grand impact en tant que système de simulation interactive à visée pédagogique. D'autres outils ont suivi ses traces avec succès, comme les applications d'enseignement de la géométrie Cabri Géomètre [CAPPONI & LABORDE 91] ou Chamois [BOURIT 00]. Avec Thinglab II, l'accent est

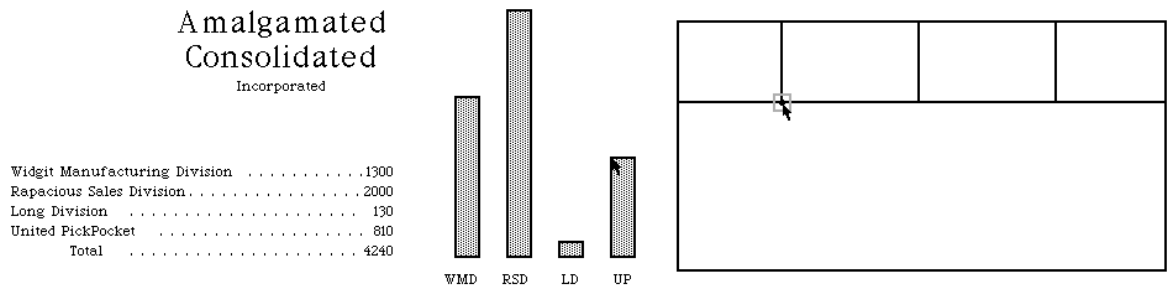


FIG. 2.32 – Deux interfaces construites avec *Thinglab*. [BORNING 79]

davantage mis sur les applications interactives réelles [MALONEY *et al.* 89]. Mais bien que des optimisations algorithmiques aient été introduites, aucun exemple réellement nouveau n'a été décrit.

L'approche mixte de Fabrik

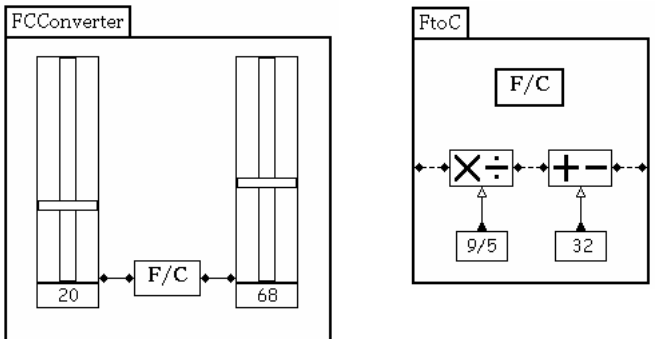


FIG. 2.33 – Un convertisseur de température construit avec *Fabrik*. [INGALLS *et al.* 88]

Fabrik [INGALLS *et al.* 88] est un système similaire à Thinglab, qui a été davantage conçu pour la description d'interfaces graphiques. Fabrik est basé sur un modèle de type data-flow compositionnel, mais qui autorise les connexions bidirectionnelles de type PPC. Certains de ses objets effectuent des calculs, d'autres sont interactifs et permettent de saisir des données (zone de texte, bouton) ou de les afficher (présentation de listes, images).

La figure 2.33 montre, à gauche, un convertisseur d'unités de température construit avec Fabrik. Celui-ci comporte deux blocs interactifs de type « barre de défilement » reliés par un bloc de calcul, défini à l'aide d'opérateurs atomiques dans la partie droite de la figure.

Les *graphèmes* constituent une autre particularité intéressante de Fabrik. Ce sont des blocs qui transforment des valeurs en objets géométriques élémentaires (lignes, rectangles) et qui permettent de décrire la présentation d'une interface. La figure 2.34 page suivante montre l'intérieur des blocs `slider` utilisés dans le convertisseur de la figure 2.33. Ce graphe comporte deux graphèmes de rectangles prenant les valeurs `topleft` et `bottomright` en entrée. Ils correspondent aux deux rectangles internes du slider, représenté à droite. La position du premier est fixe, et celle du second est calculée à partir des mouvements de la souris.

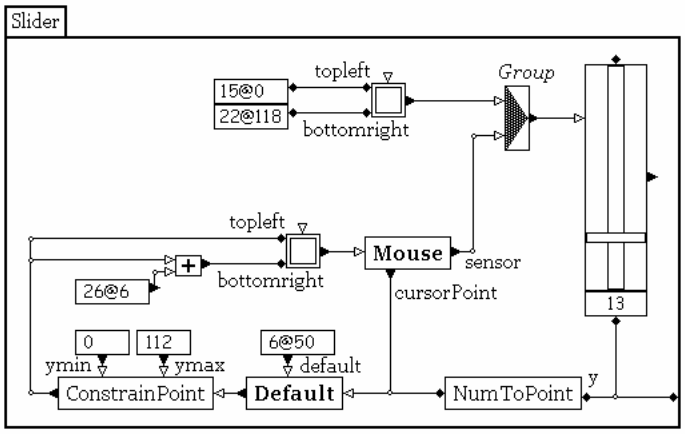


FIG. 2.34 – Un composant interactif construit avec les graphèmes de *Fabrik*. [INGALLS *et al.* 88]

Les flots de données d'InterCONS et de Whizz'Ed

Les langages visuels à flots de données ont été employés avec succès dans des domaines tels que le traitement d'images ou le traitement du son. LabView [SANTORI 90], logiciel de traitement de données provenant d'appareils de mesure, en est un exemple. Des éditeurs de ce type existent pour décrire des comportements d'interfaces graphiques.

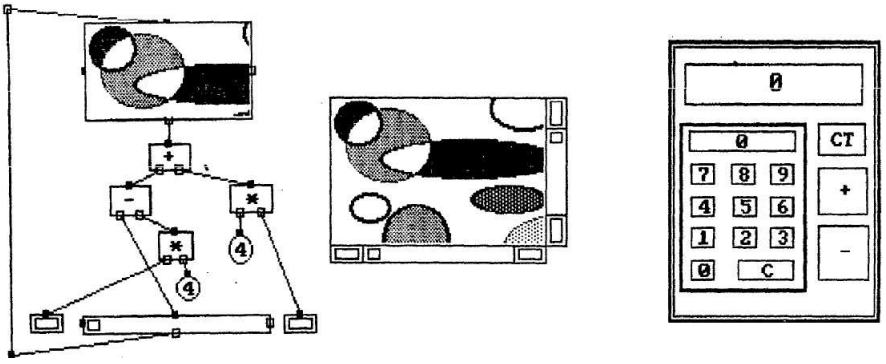


FIG. 2.35 – Un système de barres de défilement et une calculatrice construits avec InterCONS. [SMITH 88]

InterCONS [SMITH 88] est l'un de ces systèmes. Il emploie primitives graphiques appelées *outils*, qui possèdent des connecteurs en entrée et en sortie de type *entier*. Ces blocs sont des éléments interactifs ou des opérateurs simples.

La figure 2.35 montre, à gauche, une technique de défilement construite avec InterCONS. Une barre de défilement simple et deux boutons ont été connectés à une vue pour contrôler son défilement horizontal. InterCONS permet de masquer les objets comportementaux afin de ne laisser apparaître que les éléments interactifs, puis de réorganiser visuellement ces derniers : au centre de la figure 2.35, la construction a été complétée avec un contrôle vertical avant d'être visuellement remaniée. À droite de la figure est représentée une autre application, une calculatrice, construite avec InterCONS.

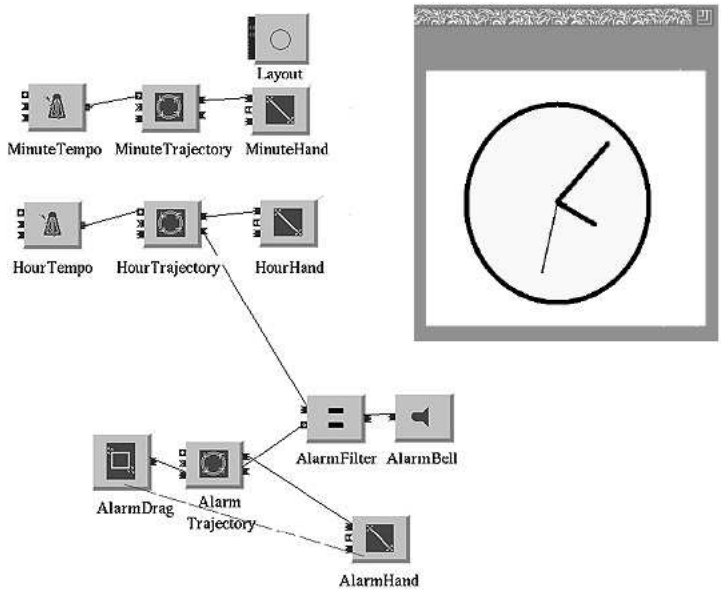


FIG. 2.36 – Une horloge analogique avec alarme réglable, décrite avec Whizz'Ed. [ESTEBAN 97].

Whizz'Ed [ESTEBAN *et al.* 95, ESTEBAN 97] est un langage visuel dédié à la construction d'interfaces à manipulation directe, qui emploie un paradigme de data-flow compositionnel. Contrairement aux éditeurs vus précédemment, la présentation de l'interface graphique est dissociée du langage visuel. Whizz'Ed comporte néanmoins des briques qui affichent des primitives graphiques, similaires aux graphèmes de Fabrik. Parmi les autres types de briques disponibles, les *tempos* émettent régulièrement des valeurs de type booléen dans le but d'effectuer des animations, et les *réactions* émettent des événements utilisateur.

La figure 2.36 illustre une horloge construite avec Whizz'Ed : L'horloge, représentée à droite de la figure, possède une aiguille d'alarme, réglable par manipulation directe, qui sonne lorsqu'elle coïncide avec l'heure courante. Cette fonction, ainsi que l'apparence de l'horloge et sa mise à jour sont décrites par le programme visuel reproduit sur la figure.

La gestion des événements dans Whizz'Ed se fait par un mécanisme d'*abonnement* : une primitive graphique est rendue sensible à un type d'événement lorsqu'elle est reliée à un bloc de réaction. Ce lien est représenté sur la figure 2.36 par des pointillés. Les réactions étant dissociées des primitives graphiques, ce mécanisme permet de rendre explicite l'interprétation des événements. De plus, de nouveaux types d'événements synthétiques peuvent être ajoutés avant d'être associés aux primitives graphiques.

Discussion sur les apports et les limites de ces démarches

La plupart des éditeurs de simulations interactives présentent l'avantage de pouvoir spécifier simultanément l'apparence et du comportement d'une interface graphique. Cependant, le comportement d'une interface peut difficilement être spécifié indépendamment de sa présentation : lorsque celui-ci change, la présentation de l'interface change également. InterCONS résout en partie ce problème en permettant de masquer certains composants graphiques.

ThingLab et ses successeurs ont montré que la PPC était bien adaptée à la description de certains mécanismes d'interaction : les comportements *géométriques* des interfaces (positionnement, proportions, alignements de widgets), et le *maintien des consistances* entre les objets de l'application et les objets de l'interface graphique ou entre des vues multiples. Il semble toutefois que ce paradigme soit beaucoup moins employé pour décrire l'interaction en entrée en tant que telle. De plus, ces outils restent dédiés aux simulations interactives utilisées dans des buts pédagogiques, plutôt qu'à la construction d'interfaces utilisateur réelles. Premièrement, ce paradigme ne permet de décrire que des relations mathématiques simples entre des objets. Ensuite, un ensemble important de contraintes est difficile à résoudre et à spécifier (système sous-contraint ou sur-contraint).

Le paradigme à flot de données semble plus prometteur pour décrire les comportements complexes des applications interactives, en particulier les techniques d'interaction. En effet, ces comportements gagnent à être décrits de manière concrète, en termes de liens de causalité entre objets [ESTEBAN *et al.* 95]. Parmi les outils existants, Whizz'Ed est actuellement l'un des plus aboutis du point de vue comportement et gestion des entrées. Bien qu'il ait été principalement conçu pour construire des animations, il a servi à décrire une application iconique complète de type Macintosh [ESTEBAN 97].

Les outils que nous avons passé en revue mettent globalement l'accent sur la construction de comportements « autonomes », qui mettent en œuvre des animations ou des calculs complexes. Certains exemples comme la calculatrice montrent qu'il est possible de créer des applications complètes, et non seulement leur partie interface. Ces applications-jouets montrent qu'il est possible de construire avec une fine granularité des composants interactifs relativement complexes.

Malheureusement, l'accent sur les comportements semble se faire au détriment de l'interaction : dans tous les exemples mis en œuvre, l'interactivité et la contrôlabilité des applications est relativement pauvre. Si les modèles de type data-flow sont naturellement adaptés à l'interaction concurrente, les outils actuels, de par leur vision des entrées, ne permettent pas une interaction riche avec des dispositifs multiples : les entrées se limitent aux événements standards. En outre, une grande partie de la gestion des entrées est implicite et non configurable.

Whizz'Ed, dédié aux applications hautement interactives, semble le plus avancé dans ce domaine. S'il est adapté à la description des interfaces à manipulation directe, sa vision de l'interaction en entrée reste cependant relativement limitée. Seuls les événements souris classiques sont pris en compte dans la version originale de Whizz, bien que son extension à l'interaction bimanuelle soit également capable d'exploiter les pointeurs multiples de façon très efficace (voir la section 2.6.2 page 70 sur cette extension). Mais le mécanisme d'abonnement de Whizz ne permet pas, sans programmation, de gérer de nouveaux types d'événements provenant de dispositifs concrets. En outre, certains mécanismes de gestion des entrées restent implicites, tels que le *picking*.

2.7.3 Les éditeurs de comportements 3D

Introduction

Certains outils comme Maya RTA [ALIAS 01], Blender [ROOSENDAAL & WARTMANN 03] et Virtools Dev [VIRTOOLS 01] permettent de créer des applications 3D hautement interactives en construisant des comportements par manipulation directe. L'objectif de ces éditeurs est de permettre aux artistes de rendre leurs créations vivantes et interactives sans nécessité de programmation.

Les éditeurs de comportements 3D sont essentiellement basés sur les paradigmes d'*association*

d'actions (mapping) ou de *flot de données*. Certaines approches plus expérimentales décrivent également des éditeurs visuels hybrides *données/contrôle*. Nous décrivons ces trois types d'éditeurs.

Les senseurs et actions de Maya RTA et Blender

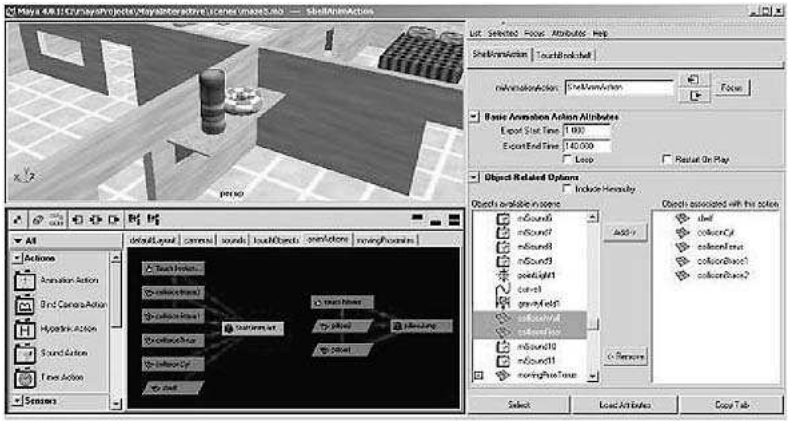


FIG. 2.37 – une fenêtre de *Maya RTA* avec son éditeur d'interaction en bas à gauche. [ALIAS 01]

Maya RTA (Maya Real-Time Author) [ALIAS 01] permet de construire des applications 3D interactives dans le format Shockwave 3D, en connectant des *senseurs* et des *actions* (figure 2.37). Les senseurs réagissent de façon binaire à des propriétés de la scène (collision, distance entre objets) ou des dispositifs d'entrée (touche du clavier, bouton de la souris). Connectées aux senseurs, les actions déclenchent des commandes de haut niveau (démarrer une animation, jouer un son, changer de caméra). Un senseur peut également déclencher plusieurs actions en parallèle ou en séquence. L'éditeur de Maya emploie un modèle événementiel binaire très simple, qui ne permet de décrire que des relations directes (*mappings*) entre des contrôleurs à deux états et des actions.

La fenêtre interactive de **Blender** [ROOSENDAAL & WARTMANN 03] emploie un modèle légèrement plus sophistiqué à base de briques logiques. Les *briques logiques* comprennent les *senseurs* et les *actuateurs* (l'équivalent des actions de Maya RTA), ainsi qu'un troisième type de brique, les *contrôleurs*. Ces derniers comprennent les opérateurs logiques et les scripts Python. Ils s'insèrent entre les senseurs et les actuateurs, et permettent de décrire des conditions plus complexes pour les déclenchements d'actions. Tout comme l'éditeur de Maya, les briques logiques de Blender sont essentiellement basées sur des *associations d'actions* où les événements sont binaires. L'accès à des données comme les coordonnées de la souris est possible, mais nécessite l'écriture de scripts.

Les flots de données de Virtools Dev

Un grand nombre d'outils de construction d'applications interactives 3D reposent sur des modèles à flot de données (voir section 2.6.2 page 75). Ce paradigme se prêtant bien à une programmation visuelle, certains d'entre eux proposent en complément un éditeur interactif. C'est le cas par exemple de WorldUp [SENSE8 03], Cult3D Designer [CYCORE 03] ou Virtools Dev [VIRTOOLS 01].

Le modèle à base de flots de données de l'éditeur de **Virtools Dev** [VIRTOOLS 01] est parmi les

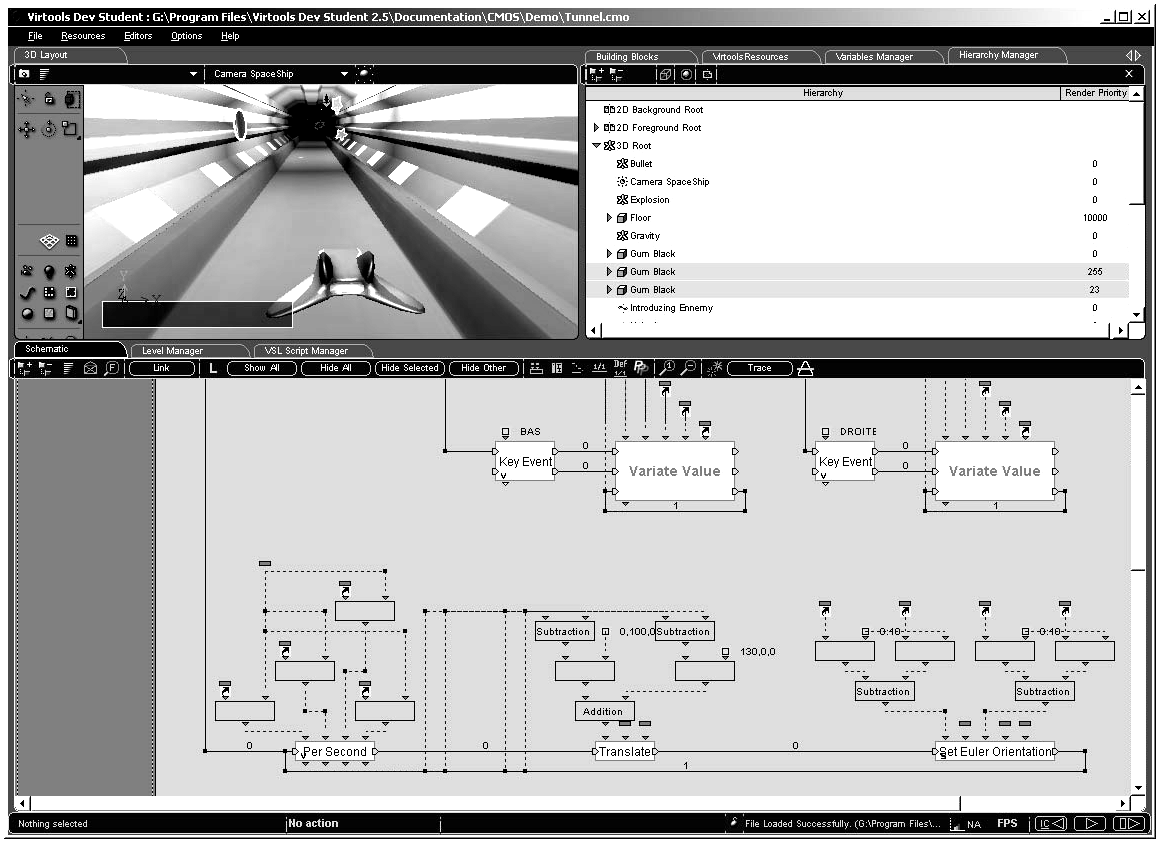


FIG. 2.38 – Un jeu d'action programmé avec l'éditeur de comportements de *Virtools Dev*.

plus sophistiqués sur le marché. Virtools comporte un grand nombre de scripts comportementaux prédéfinis, graphiquement représentés par des *blocs de construction*. Associé à un objet 3D, un script comportemental permet par exemple d'animer ou de contraindre ses mouvements. Tout comme les briques logiques, chaque bloc comprend des entrées et des sorties d'activation (contrôle), mais peut également traiter des données typées à travers des *paramètres d'entrée* et *paramètres de sortie*. Les blocs peuvent être librement interconnectés à travers leurs entrées/sorties et leurs paramètres pour décrire des comportements complexes. Les blocs « contrôleur » (souris, clavier, joystick) permettent d'ajouter de l'interaction. Les blocs sont composables, et de nouveaux blocs atomiques peuvent être définis sous forme de scripts en C^{++} ou de DLLs.

La figure 2.38 page précédente montre un jeu d'action programmé avec Virtools. La fenêtre de jeu est visible dans la partie supérieure gauche de la fenêtre, la structure de la scène apparaît dans la partie supérieure droite, et une partie du script décrivant le contrôle du vaisseau est visualisée en bas. Ce script emploie quatre blocs `Key Event` (gauche, droite, bas, haut) dont deux sont visibles ici. Chacun de ces blocs est connecté à un bloc `Variate Value` qui, à partir des paramètres de déplacement du vaisseau (vitesse, accélération, direction), émet des messages vers la partie inférieure du script qui se charge de calculer puis d'appliquer les transformations géométriques sur l'objet.

Flots de données et machines à états de VRED

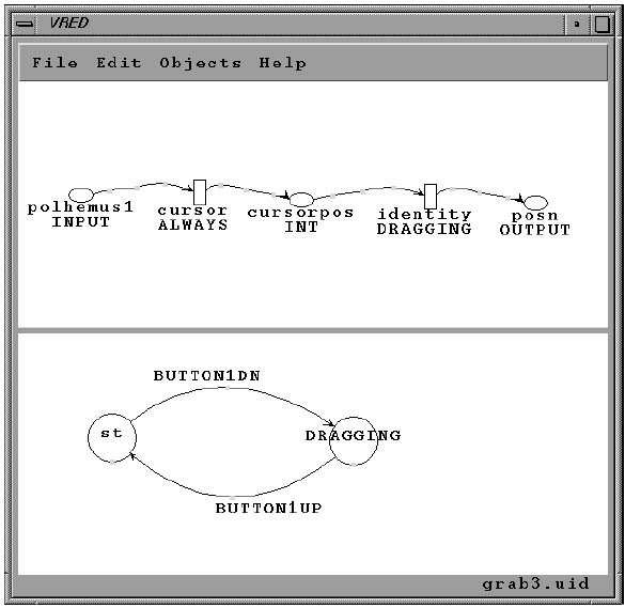


FIG. 2.39 – Le déplacement d'un objet 3D décrit avec VRED [JACOB *et al.* 99].

VRED [JACOB *et al.* 99] est un éditeur d'interaction expérimental qui s'appuie sur un modèle mixte à flot de données et à flot de contrôle pour décrire les aspects à la fois continus et discrets de l'interaction. Son objectif est d'incorporer les deux aspects dans le même langage visuel dans le but de décrire facilement des techniques « Non-WIMP ». Il a été principalement employé pour spécifier des techniques d'interaction simples de réalité virtuelle.

VRED se compose d'un *éditeur de flot de données* pour décrire des relations entre des grandeurs continues, et d'un *éditeur de diagrammes de transition d'états* (voir section 2.3.1 page 44) pour décrire des modes. Chaque mode active des liens et en désactive d'autres dans le flot de données.

La figure 2.39 page ci-contre illustre une technique simple de manipulation d'objets dans une application de réalité virtuelle, où une main dont la position est captée par un Polhemus déplace des objets lorsqu'un bouton est maintenu enfoncé (le picking n'est pas représenté). Dans le flot de données (en haut), les ellipses représentent des valeurs continues (avec en-dessous leur type) et les rectangles des transformations (avec en-dessous leurs conditions d'activation). La position fournie par le Polhemus est liée à celle d'un curseur, elle-même liée à celle de l'objet. Le premier lien est statique et le le second est uniquement actif dans le mode *DRAGGING*. La prise en charge des événements discrets de type bouton est décrite par un diagramme de transition d'états (en bas), qui active le mode *DRAGGING* tant que le bouton est enfoncé.

Discussion sur les apports et les limites des éditeurs 3D

Les modèles à base de senseurs et de canaux des éditeurs 3D offrent une vision détaillée des entrées, qui permet d'exploiter au mieux les dispositifs étendus. Ils sont par conséquent beaucoup plus ouverts aux dispositifs non-standards que les outils d'édition 2D, et permettent de décrire des interactions qui emploient des entrées extrêmement riches. Les modèles de connexion naïfs comme ceux de Maya RTA et de Blender offrent par contre un pouvoir d'expression assez faible, qui ne permet pas de décrire des comportements complexes.

Les éditeurs de type VirTools Dev ont comblé ce manque en exploitant le paradigme de data-flow. VirTools Dev, de par son grand pouvoir d'expression, a pu servir à développer des jeux vidéo commerciaux tels que Syberia [MICROÏDS 02]. D'autres outils 3D emploient des éditeurs de comportements similaires [SENSE8 03, CYCORE 03]. Le principal inconvénient de ces outils est leur complexité d'utilisation qui requiert des compétences qui sont proches de celles du programmeur.

Les éditeurs 3D du commerce sont uniquement adaptés à la description d'interfaces 3D, où les attributs géométriques des objets d'une scène sont contrôlés plus ou moins directement par l'utilisateur. Si ce type d'interface est très stéréotypé, les éditeurs 3D sont peu adaptés au contrôle d'applications 2D, où les objets du domaine et les techniques d'interaction sont plus variés et souvent plus complexes.

VRED est une approche expérimentale qui constitue l'une des rares tentatives visant à réconcilier les aspects flot de données et les rassembler dans le même langage visuel. Cependant, il emploie deux notations bien différentes et oblige pour chaque aspect de l'interaction à opter pour un modèle continu ou à événements. Les difficultés liées à l'interprétation d'un flot de données dynamique ont en outre été évoquées par l'auteur lui-même, qui propose d'encapsuler un flot de données dans chaque état du diagramme de transition et d'étendre l'éditeur avec un paradigme d'interfaces zoomables. Outre les difficultés de lecture que cette nouvelle approche peut poser, cet éditeur n'a pas encore vu le jour.

2.7.4 Conclusion

Les éditeurs graphiques d'interaction constituent un pas en avant non négligeable vers des interfaces graphiques configurables en entrée. Malheureusement, il existe encore une trop grande séparation entre les éditeurs 2D qui sont puissants mais possèdent une vision événementielle stéréotypée des entrées, et les éditeurs 3D qui ont une vision générique et bas-niveau des dispositifs d'entrée mais

ne permettent pas de décrire des techniques d'interaction complexes. En outre, les premiers (excepté Whizz'Ed) reposent en général sur des modèles à base de contraintes, ce qui les rend plus adaptés à la description de comportements géométriques qu'à des flux d'entrée.

Nous remarquons également que les éditeurs sont soit "orientés-édition", soit "orientés-modèle". La première approche consiste, comme Maya RTA, à construire un éditeur interactif destiné à l'utilisateur final employant des métaphores simples, mais reposant sur un modèle simpliste de l'interaction, avec un faible pouvoir d'expression. La seconde approche consiste, comme VRED, à bâtir un éditeur visuel par-dessus un modèle existant, en général puissant, mais sans se focaliser sur l'utilisabilité et sans prendre le temps d'implémenter les techniques d'interaction et de visualisation appropriées.

Pour finir, les outils d'édition 2D comme 3D semblent tout ignorer des techniques d'interaction avancées telles que l'interaction gestuelle, les outils transparents ou la reconnaissance vocale. Un outil de configuration de l'interaction en entrée gagnerait à s'inspirer des approches employées dans les deux domaines, tout en fournissant un support pour ces techniques d'interaction non standards.

2.8 Synthèse

Vu la grande complexité des applications interactives, des modèles ont été rapidement proposés afin de séparer les différents aspects fonctionnels et structurels des interfaces, et de guider leur conception et leur implémentation. Les premières approches de type vertical comme *Seeheim* ou *Arch* décomposent l'interface en plusieurs couches d'abstraction, de l'utilisateur jusqu'au noyau fonctionnel. Malgré leurs apports théoriques, ces modèles sont peu détaillés et encapsulent l'interaction en entrée et en sortie bas-niveau dans une seule entité monolithique. Afin de rendre mieux compte des caractéristiques des interfaces modernes et du paradigme de programmation orientée-objet, les approches horizontales « à agents » comme *MVC*, *PAC* et les *interacteurs formels* décomposent les interfaces en réseaux d'entités de même nature. Les entrées, dispersées parmi les agents, restent cependant confinées dans des objets monolithiques et ne sont toujours pas décrites. Elles restent en outre confondues avec les sorties, hormis dans le modèle *MVC* qui sépare les entrées des aspects graphiques mais sans expliquer comment mettre en œuvre cette séparation. Le modèle des *interacteurs de CNUCE* suggère néanmoins des flux d'entrée qui transitent d'un agent à l'autre pour être converties en données abstraites interprétables par l'application.

Deux types de modèles se sont spécifiquement attachés à décrire l'interaction en entrée bas-niveau. Les modèles logiques de dispositifs comme *GKS* et *PHIGS*, dont l'objectif est l'indépendance vis-à-vis du matériel, divisent les dispositifs d'entrée en six classes. Les *tâches d'interaction de Foley* séparent de façon similaire les tâches applicatives en six classes génériques. Malgré leurs apports pratiques, et en dehors du fait qu'elles ont été construites pour des besoins d'interaction qui ne sont plus les mêmes aujourd'hui, ces classifications sont excessivement simplificatrices au vu de la grande variété des dispositifs (pour les premières) et des tâches existantes (pour les secondes). Les modèles physiques de *Buxton* et *Card* mettent au contraire l'accent sur la spécificité de chaque dispositif. Le premier propose une taxonomie basée sur des caractéristiques pragmatiques (nombre de dimensions, propriétés physiques captées) et le second décrit un dispositif d'entrée comme une composition de dispositifs élémentaires qui convertissent des grandeurs physiques en valeurs numériques. Ces modèles sont utiles pour comparer différents dispositifs ou en concevoir de nouveaux, mais n'ont eu que peu d'impact sur les applications et les outils de développement, où l'on recherche surtout la généricité et la portabilité. Seules les librairies d'accès bas-niveau aux dispositifs emploient des représentations de

nature physique, où chaque dispositif concret est habituellement exposé comme une combinaison de dimensions continues et discrètes.

Aujourd'hui, le développement des applications interactives repose exclusivement sur les boîtes à outils graphiques, qui gèrent la majeure partie de l'interaction en entrée de façon interne. Les boîtes à outils conventionnelles, câblées pour une utilisation exclusive et stéréotypée de la souris et du clavier, sont les principales responsables de la rigidité des applications en termes d'entrées. Des boîtes à outils WIMP avancées comme *Subarctic* et *Garnet/Amulet* ont été proposées afin de décrire la gestion des entrées et l'interaction standard de manière claire et extensible, mais l'intégration de nouvelles techniques d'interaction nécessite en pratique des remaniements importants, et les dispositifs non-standard sont ignorés. Un certain nombre de boîtes à outils plus spécialisées prennent en charge des paradigmes d'interaction Post-WIMP spécifiques, telles les boîtes à outils gestuelles (*Satin*), parfois en exploitant des entrées non-standard comme l'extension multimodale de *Subarctic* (entrées de nature ambiguë) ou les boîtes à outils multi-pointeurs (*MMM*, *Bimanual Whizz* et *MID*). Le modèle d'interaction à base d'outils transparents a également donné lieu à un remaniement de la boîte à outils *MMM* et au développement de l'architecture *CPN2000*. Ces boîtes à outils emploient toutes cependant un modèle à événements conventionnel adapté à des dispositifs de type spécifique, et exploitent ces dispositifs de manière efficace mais figée.

Les boîtes à outils spécialisées dans l'interaction tangible (*Phidgets*) et la sensibilité au contexte (*Context Toolkit*), peu nombreuses vu la jeunesse de ces paradigmes, offrent une prise en charge souple mais de bas-niveau de senseurs et dispositifs simples de nature hétérogène. Le domaine de l'interaction 3D, plus mûr, offre depuis un certain temps des boîtes à outils permettant de connecter librement des canaux individuels de dispositifs d'entrée sophistiqués à des structures abstraites, habituellement à travers des flots de données (*UGA*, *Inventor*, *TBAG*, *WorldToolkit*). Ces structures sont cependant toujours des attributs d'objets 3D, les dispositifs pris en charge sont tous des périphériques 3D, et les techniques d'interaction qu'il est possible de décrire sont bien moins variées que celles de la 2D. Le modèle à flot de données sur lequel elles reposent offre néanmoins une grande liberté d'expression et se prêtent bien à des outils d'édition visuelle (*Maya RTA*, *Blender*, *Virtools Dev*) qui rendent la spécification de l'interaction en entrée en partie accessible à des non-programmeurs.

Le domaine de l'interaction 2D a également produit des éditeurs de comportements, mais qui reposent pour la plus grande part sur des modèles à contraintes peu adaptés à la description de l'interaction en entrée bas-niveau (*Thinglab*, *Fabrik*). Seul l'éditeur *Whizz'Ed* a su exploiter la grande souplesse offerte par les flots de données, initialement pour construire des interfaces animées, puis pour décrire certains mécanismes propres à l'interaction bimanuelle. Notons que des systèmes à transitions (automates à états finis, réseaux de Petri) ont été également employés pour décrire l'interaction de manière visuelle ou non : *Petshop*, par exemple, permet de construire interactivement des spécifications *ICO* à base de réseaux de Petri. Cependant, les systèmes orientés-contrôle que ces formalismes se proposent de décrire caractérisent mieux les techniques WIMP que les interactions Post-WIMP, essentiellement amodales et fortement concurrentes.

Pour résumer simplement, aucun modèle et aucun outil à ce jour ne prend en compte l'adaptabilité en entrée sous ses trois angles. Deux types d'approche ont été proposées pour améliorer la contrôlabilité : celle des boîtes à outils Post-WIMP, qui consiste à implémenter un modèle d'interaction efficace pour un certain type d'entrées, et celle des boîtes à outils 3D, qui consiste à profiter de manière simple de la richesse des dispositifs à dimensions multiples. La voie vers la configurabilité a été ouverte par les éditeurs d'interaction 2D et surtout 3D. Aucun de ces éditeurs, cependant, ne permet d'exploiter des techniques d'interaction Post-WIMP sophistiquées, ni de tirer parti de dispositifs multiples de

nature hétérogène (par exemple, geste, parole et dispositifs 3D). Enfin, ni les outils de programmation ni les éditeurs interactifs n'offrent de solution adaptée à l'interaction avec des entrées appauvries.

Le problème de l'adaptabilité en entrée peut néanmoins être résolu en approfondissant les approches existantes et en tirant parti de leurs avantages respectifs. Dans le chapitre suivant, nous introduisons une métaphore pour décrire des interfaces intégralement configurables en entrée, et nous verrons comment cette métaphore mène tout naturellement à un modèle à flot de données, similaire à ceux des outils 3D ou de *Whizz'Ed*. Nous montrerons également en quoi les particularités de notre modèle et de nos outils en font une solution particulièrement efficace au problème de l'adaptabilité en entrée.

Chapitre 3

Introduction au modèle des configurations d'entrée

Sommaire

3.1 Introduction

L'adaptabilité en entrée requiert une grande indépendance entre les dispositifs physiques et les applications. Notre objectif premier est par conséquent de découpler les dispositifs d'entrée de l'application, afin que le développeur ou l'utilisateur puisse employer n'importe quel dispositif d'entrée *à la place* des dispositifs standards, ou *en plus* de ceux-ci. Sur un système entièrement adaptable en entrée, celui-ci devrait idéalement disposer d'un outil lui permettant de *connecter (et déconnecter) librement ses dispositifs d'entrée à l'application*.

Nous avons choisi de baser notre approche sur ce principe de connexion, car il constitue une métaphore simple et naturelle aussi bien pour l'utilisateur que pour le développeur.

Dans un premier temps, nous motivons et développons cette métaphore pour aboutir à un *paradigme de dispositifs en cascade*, qui constituera la base de notre approche. Dans un deuxième temps, nous présentons brièvement le paradigme des *systèmes réactifs* qui a inspiré notre modèle d'exécution. Enfin, nous présentons notre modèle des *configurations d'entrées*, qui concrétise et combine ces deux approches.

3.2 Le paradigme des dispositifs en cascade

Dans cette section, nous introduisons le paradigme des dispositifs en cascade, en développant le principe de la connexion logicielle de dispositifs. Ce paradigme suppose d'abord deux types d'entités : les *dispositifs d'entrée* physiques et l'*application*. Nous serons par la suite amenés à introduire quatre autres notions essentielles : les *points d'entrée*, les *adaptateurs*, les *dispositifs généralisés*, et enfin les *cascades*.

3.2.1 La métaphore de la connexion logicielle

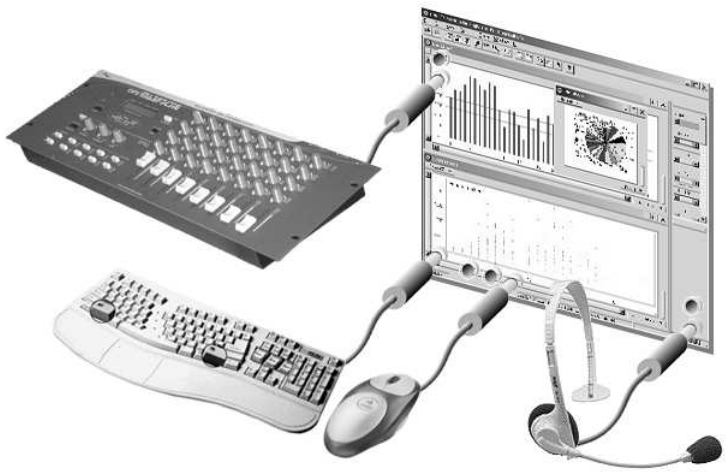


FIG. 3.1 – La métaphore de la connexion : les dispositifs d'entrée sont connectés à une application pour être utilisés.

Le paradigme de connexion *logicielle* de dispositifs (figure 3.1 page précédente) dérive de la notion plus familière de connexion *physique* : lorsque nous voulons utiliser un dispositif, nous le *connectons*. Dans le monde quotidien, la plupart des dispositifs électriques ou électroniques nécessitent d'être branchés sur le secteur (fer à repasser, chargeur de téléphone portable), ou sur un autre dispositif (casque audio sur une chaîne hi-fi, guitare électrique sur un amplificateur) pour être utilisés. Dans le deuxième cas, la connexion sert à établir un échange d'informations entre un dispositif mobile à usage humain, et un dispositif fixe qui lui est lié.

Il est intéressant de noter que lorsque l'échange d'informations se fait du dispositif humain vers le dispositif fixe, il est possible d'établir un parallèle avec l'interaction instrumentale dans le monde réel [BEAUDOUIN-LAFON 97]. Dans ce cas particulier de manipulation instrumentale, l'opération de connexion *initie* l'utilisation d'un instrument (le dispositif mobile) tout en *désignant* l'objet du monde sur lequel il va agir (le dispositif fixe). C'est le cas lorsque l'on branche une guitare sur un amplificateur, ou un dispositif de commande sur certains appareils industriels ou domotiques. Mais cela peut également s'appliquer lorsque l'on branche un dispositif d'entrée sur un ordinateur (d'autant plus que les progrès technologiques autorisent dorénavant des connexions à chaud de dispositifs d'entrée [USB/HID 01]).

FIG. 3.2 – Façade d'un amplificateur de guitare électrique (1959SLP de chez Marshall) comportant quatre entrées distinctes. Chaque entrée contrôle un type de son particulier en termes de gain et de brillance.

En outre, différents branchements possibles sur un dispositif fixe peuvent multiplier le nombre d'objets manipulables. Les différentes entrées de certains amplificateurs de guitare permettent ainsi de manipuler plusieurs sons (figure 3.2). Mais malheureusement, les systèmes informatiques actuels ne permettent qu'une utilisation unique et globale d'un dispositif (voire zéro, s'il n'est pas reconnu par les applications). Le paradigme de connexion logicielle permet d'aller plus loin dans ce domaine, en passant virtuellement d'un objet manipulable unique (l'ordinateur) à des objets multiples de granularité plus fine (les applications et leurs différentes fonctionnalités). La connexion logicielle *étend* la connexion physique des dispositifs d'entrée, tout en se basant sur le même paradigme.

3.2.2 Points d'entrée

Tout logiciel interactif possède un certain nombre de besoins en termes d'entrée, ou *points d'entrée* : ce sont des données qui sont fournies par l'utilisateur. Il existe plusieurs manières de décrire ces points d'entrée indépendamment des dispositifs. L'une de celles-ci, que nous utiliserons temporairement pour les besoins de cette introduction, consiste à les identifier comme autant de *tâches d'interaction* au sens de Foley [FOLEY *et al.* 84]. Pour rappel, ces tâches d'interaction sont essentiellement des types de données.

Ainsi, un logiciel de traitement de texte pourra avoir besoin de données de type `Position` pour spécifier l'emplacement du point d'insertion, et de données de type `Text` pour insérer du nouveau texte. Il comportera ainsi deux points d'entrée. Si le logiciel gère les annotations, il pourra également

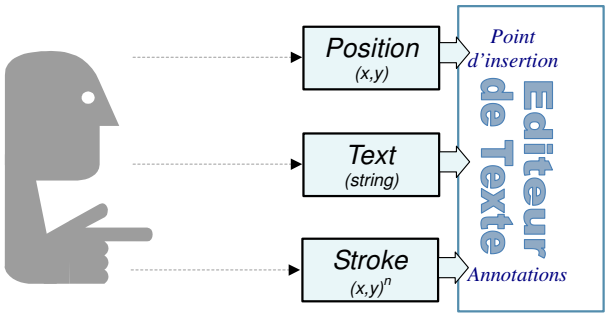


FIG. 3.3 – Les trois points d'entrée de l'application de traitement de texte, décrits comme des tâches d'interaction de Foley.

déclarer un point d'entrée supplémentaire de type `Stroke` (figure 3.3).

Chaque point d'entrée définit une destination de connexion possible pour un dispositif d'entrée. Même si dans les applications classiques les points d'entrée de type `Text` et `Position` sont invariablement connectés à un clavier et à une souris, les tâches d'interaction sont théoriquement indépendantes des dispositifs d'entrée. Dans la section suivante, nous verrons quelques exemples de connexions standards et alternatives.

3.2.3 Adaptateurs

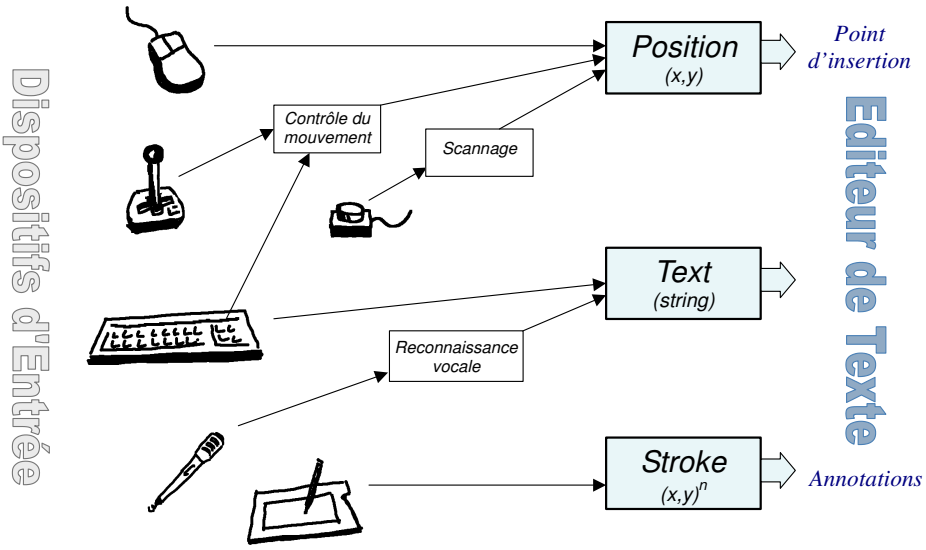


FIG. 3.4 – Différentes manières de connecter des dispositifs d'entrée à une application de traitement de texte.

La figure 3.4 illustre, à l'aide de connexions, diverses méthodes de saisie possibles pour notre logiciel de traitement de texte. Du côté gauche sont représentés un certain nombre de dispositifs d'entrée

potentiellement utilisables (souris, manette, bouton à deux états, clavier, microphone, tablette) et du côté droit les trois points d'entrée (tâches d'interaction de Foley) vers l'application. Seules quelques-unes parmi toute les possibilités de connexion sont représentées sur la figure.

Certaines connexions font intervenir des utilisations non triviales des dispositifs d'entrée (par exemple, émettre des positions avec un clavier). Les données émises par le dispositif et celles reçues par le point d'entrée étant incompatibles, ces couplages nécessitent l'utilisation de boîtes adaptatrices, ou *adaptateurs*. Chaque adaptateur traite les données en provenance d'un dispositif pour les rendre compatibles avec un point d'entrée donné. Ils peuvent éventuellement posséder un état et produire des retours (graphique, sonore, etc...) qui ne sont pas représentés sur la figure.

Il est intéressant de remarquer que chacun de ces adaptateurs décrit une *technique d'interaction*. Ainsi, l'utilisation d'une manette ou d'un clavier comme dispositifs positionnels implique l'utilisation d'une technique de *contrôle du mouvement* ou contrôle du second ordre (l'utilisateur contrôle la vitesse et non la position du pointeur). Des techniques d'animation (dites de *scannage*) sont utilisées pour spécifier des positions avec des dispositifs d'accessibilité à deux états, tels que les boutons. De même, pour pouvoir fournir du texte, le signal d'un microphone nécessite d'être interprété à travers un module de reconnaissance vocale.

Sur la figure, par souci de simplicité, les connexions standards ne comportent pas d'adaptateurs. En pratique, certaines d'entre elles peuvent nécessiter des traitements de base (telles que des remises à l'échelle pour les dispositifs de pointage) ou un retour graphique simple (pointeur, par exemple). Les données brutes du clavier nécessitent également d'être traitées (le cas du clavier sera développé un peu plus loin).

3.2.4 Dispositifs généralisés

Nous avons introduit trois types d'objets pour décrire des exemples pratiques de connexion de dispositifs : les dispositifs d'entrée, les adaptateurs, et les points d'entrée vers l'application. Nous allons unifier ces trois entités.

Remarquons d'abord qu'il est possible de connecter deux types d'objets à un point d'entrée : un dispositif ou un adaptateur. Or du point de vue de l'application, un adaptateur se comporte exactement comme un dispositif d'entrée. Par exemple, l'adaptateur "contrôle du mouvement" fournit des données comparables à celles d'une souris.

Notons également qu'un point d'entrée peut également se comporter comme un dispositif d'entrée. Ainsi, des positions peuvent générer du texte à travers un clavier graphique. Des tracés peuvent également produire du texte à travers un module de reconnaissance de l'écriture (figure 3.5 page suivante).

Chaque dispositif d'entrée, adaptateur et point d'entrée peut jouer le rôle de dispositif. Nous regroupons par conséquent ces objets sous le terme générique de *dispositifs généralisés*, ou plus simplement *dispositifs*. Les *dispositifs système* tels que le clavier ou la souris, sont des cas particuliers de dispositifs. Les points d'entrée vers une application seront également appelés *dispositifs d'application*.

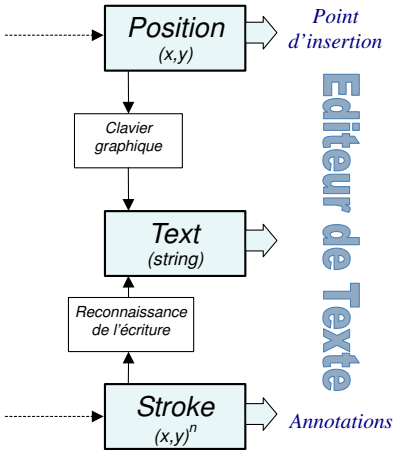


FIG. 3.5 – Exemples de points d'entrée jouant le rôle de dispositifs

3.2.5 Cascades

Nous avons vu jusqu'ici deux types de connexions : les connexions directes et les connexions avec adaptateur unique. Dans les systèmes interactifs cependant, l'interprétation des actions de l'utilisateur se fait en général à travers une série de traitements successifs. Cette sérialisation répond souvent au besoin d'accéder à un dispositif sur plusieurs niveaux d'abstraction. Elle apporte en outre l'avantage d'une modularité accrue et d'une meilleure structuration. Le mécanisme de gestion du clavier, que nous étudierons dans cette section, fournit un exemple de traitement en cascade.

L'exemple du clavier

A chaque fois qu'une touche d'un clavier est appuyée, on peut considérer qu'elle passe d'un état *faux* à un état *vrai*. Ces changements d'états sont traduits par le clavier en *codes touches*, qui à leur tour sont convertis en *symboles* dont la combinaison est interprétée comme une *chaîne de caractères* dans l'éditeur de texte.

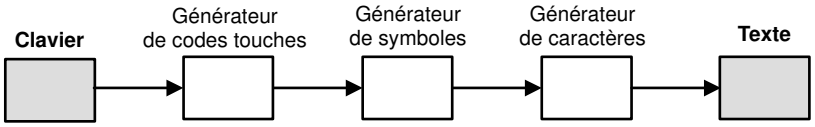


FIG. 3.6 – Cascade de dispositifs entre le clavier et l'éditeur de texte

Cette série de traitements peut être représentée par une chaîne de dispositifs allant du dispositif concret au dispositif de l'application (figure 3.6) [FEKETE & DRAGICEVIC 00]. Chaque adaptateur convertit les données qu'il reçoit dans son format propre. Le générateur de codes touches associe à chaque touche du clavier un code positionnel. Le générateur de symboles traduit les codes touches en symboles, selon la langue du clavier. Enfin, le générateur de caractères traduit les symboles en chaînes de caractères, selon une méthode d'entrée spécifique. Il permet par exemple de composer des lettres

accentuées telles que Ê, ou de produire n'importe quel caractère spécial en utilisant la touche `Alt` et les touches du pavé numérique.

Le feedback en cascade

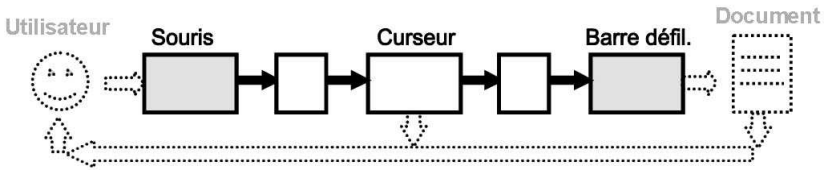


FIG. 3.7 – Cascade de dispositifs et de feedback dans une technique de défilement de document.

La figure 3.7 illustre un exemple de traitement en cascade dans une technique de défilement de document : les données en provenance d'un dispositif *souris* concret sont converties en coordonnées écran pour contrôler un dispositif *curseur*, puis encore traitées pour déplacer une barre de défilement. Ici, l'ajout de deux dispositifs « virtuels », l'*utilisateur* et le *document* permet de mettre en évidence un flux d'informations implicite : l'utilisateur contrôle la souris, le curseur émet des informations visuelles en destination de l'utilisateur, et la barre de défilement (ici, le point d'entrée vers l'application) manipule un document (caché dans l'application) qui émet à son tour des informations visuelles vers l'utilisateur.

Le flux d'informations implicite mis ici en évidence expose différents niveaux de *feedback* vers l'utilisateur. Il montre comment à une *cascade explicite de dispositifs* peut correspondre une *cascade implicite de feedback* orientée en sens inverse.

3.3 Les systèmes réactifs

Le paradigme des systèmes réactifs nous vient des langages comme Esterel [BERRY *et al.* 87, BERRY 99] ou Lustre [HALBWACHS *et al.* 91], dans lesquels la propagation de l'information est conceptuellement instantanée. Dans cette section, nous décrivons brièvement ce paradigme et motivons nos choix quant à ce modèle d'exécution.

3.3.1 Systèmes réactifs et conversationnels

Les systèmes informatiques qui interagissent avec l'environnement peuvent être distingués en deux classes [BERRY 89] :

– *Les systèmes conversationnels* [1] *:* Dans ces systèmes, le client émet des requêtes vers l'ordinateur. Celui-ci choisit s'il doit ou non y répondre, ainsi que le moment où il doit fournir la réponse. Dans ces systèmes, l'ordinateur est maître de l'interaction, et le client attend d'être servi. D'autre part, les choix internes de l'ordinateur font que ces systèmes sont le plus souvent

[1] Dans la publication originale, Berry qualifie ces systèmes d'*interactifs*. Pour éviter toute confusion avec notre notion plus générale de systèmes interactifs, nous avons choisi d'utiliser à la place le terme *conversationnel*.

non déterministes. Les exemples incluent les systèmes d'exploitation, les bases de données, les systèmes réseau.

– *Les systèmes réactifs* ou *systèmes réflexes :* Dans ces systèmes, c'est l'environnement qui est maître de l'interaction : le rôle de l'ordinateur est de réagir de façon continue aux stimuli externes en produisant des réponses instantanées. On trouve ces systèmes dans les domaines où le client ne peut pas attendre (systèmes temps réel), comme le contrôle de processus industriels, les systèmes embarqués, les pilotes de dispositifs, ou le traitement de signal. Le déterminisme est une propriété souvent désirable pour ces types de systèmes : les résultats émis doivent dépendre uniquement des informations reçues et de leur enchaînement temporel.

Les systèmes réactifs sont des programmes qui réagissent à l'environnement à la vitesse de l'environnement, alors que les autres systèmes réagissent à leur vitesse propre. Les systèmes conversationnels sont asynchrones et souvent non-déterministes, et les systèmes réactifs sont parfaitement synchrones et souvent déterministes [BERRY 00].

3.3.2 Pour une gestion réactive de l'interaction

La plupart des systèmes comportent à la fois des éléments conversationnels et réactifs. Par exemple, le pilotage d'un avion est réactif, alors que la communication avec le sol est conversationnel. C'est le cas également des systèmes interactifs.

Certains services du système d'exploitation et du système de fenêtrage, comme la gestion du curseur de la souris ou le déplacement des fenêtres, sont éminemment réactifs. Au niveau de l'application, il existe également des éléments plus réactifs que d'autres. Dans une application de dessin vectoriel, par exemple, le réaffichage d'un dessin complexe peut prendre du temps, et s'effectue selon un mode conversationnel. Il est par contre souhaitable que le retour graphique associé aux déplacements d'objets soit réactif.

Le modèle à événements classique, bien qu'essentiellement conversationnel, est souvent utilisé pour décrire l'intégralité du comportement d'une application interactive. Nous pensons cependant que les parties intrinsèquement réactives d'une application interactive, en particulier celles relatives à la gestion des entrées, gagnent à reposer sur un modèle réactif. La file à événements n'y est pas nécessaire, elle est peu adaptée à la description de comportements réactifs, et ne donne aucune garantie en termes de temps réel.

3.3.3 Le modèle du synchronisme parfait

Les langages réactifs sont basés sur le modèle du synchronisme parfait, dans lequel les processus sont capables, au niveau conceptuel, de s'exécuter et d'échanger de l'information en un temps nul [BERRY 00]. Ce modèle est équivalent au modèle 'zéro-délai' des circuits électroniques (figure 3.8 page ci-contre). Bien que ce paradigme soit peu naturel pour des programmeurs classiques, c'est un standard pour les théoriciens du contrôle, les concepteurs de circuits électroniques, ou les utilisateurs de contrôleurs programmables.

Dans le paradigme réactif, un événement est un *signal*. Du point de vue logiciel, un programme réactif s'exécute en une séquence ininterrompue d'itérations appelées *ticks*, qui sont des réactions aux signaux d'entrée. L'hypothèse du synchronisme parfait est vérifiée en pratique si les réactions sont *assez rapides* pour qu'aucun signal provenant de l'environnement ne soit perdu.

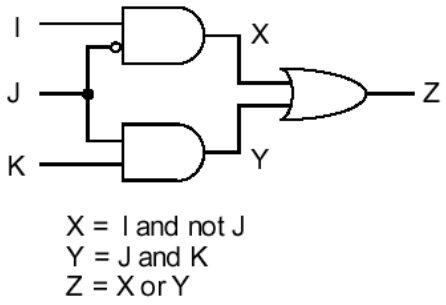


FIG. 3.8 – Exemple de circuit synchrone, où l'information est propagée de façon instantanée [BERRY 00].

Nous n'aborderons pas les détails concernant les différents langages réactifs existants (pour plus de détails, le lecteur est invité à lire [BERRY 00]). Précisons simplement qu'il en existe deux styles : les langages impératifs et les langages à flots de données. Notre modèle d'exécution s'inspire du dernier.

3.4 Les configurations d'entrée

Le modèle des *configurations d'entrée* est une concrétisation du paradigme des *dispositifs en cascades* et des systèmes réactifs décrits précédemment. La boîte à outils ICON et son langage visuel, décrits dans le prochain chapitre, constituent une implémentation de ce modèle. Nous présenterons ici le modèle des configurations d'entrée, ses différentes briques de base et ses caractéristiques principales.

Le modèle des configurations d'entrée repose sur une infrastructure et des mécanismes concrets, qui consistent principalement en un système à flots de données et un moteur réactif. Cette infrastructure et ces mécanismes sont décrits par le modèle ICOM (Input Configuration Model), indépendamment des aspects graphiques et architecturaux du modèle que nous décrivons ici. ICOM, détaillé en annexe, définit les aspects statiques (la *structure*) et dynamiques (le comportement en *édition* et en *exécution*) d'une configuration d'entrée. Nous ne donnerons ici qu'un aperçu général de son fonctionnement.

3.4.1 Les briques de base

Les quatres entités principales de notre modèle sont les configurations d'entrée, les dispositifs, les slots et les connexions.

Une **configuration d'entrée** (figure 3.9 page suivante) est un ensemble de dispositifs *système* et de dispositifs d'*application* reliés par des dispositifs *utilitaires*. Les premiers sont des objets *fournis par le système*, qui sont partagés et qui représentent pour la plupart des dispositifs d'entrée concrets. Les seconds sont des objets spécifiques *fournis par l'application*. Les derniers sont des objets autonomes *fournis par une bibliothèque* et qui sont instanciables a volonté.

Un **dispositif** (figure 3.10 page suivante) est une boîte noire qui interagit avec son environnement par l'intermédiaire de canaux typés, appelés *slots*. Sa fonction principale est de produire des valeurs

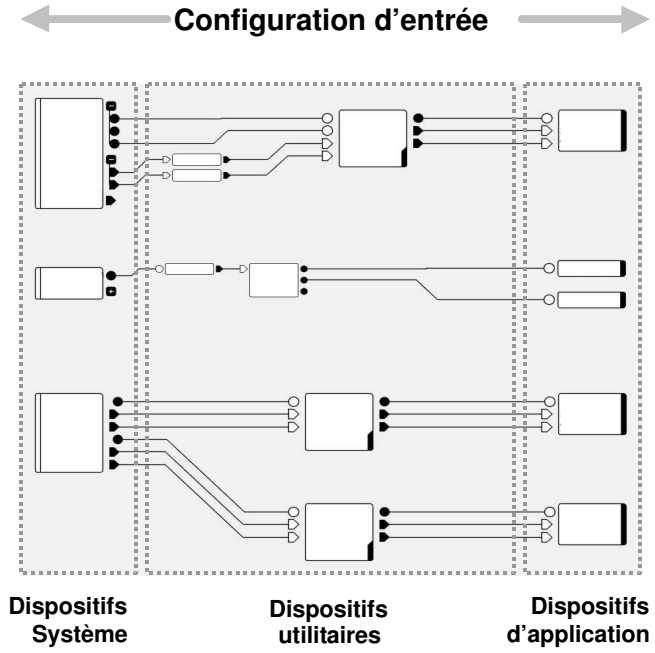


FIG. 3.9 – Une configuration d'entrée décrit une manière de relier des dispositifs système à des dispositifs d'application.

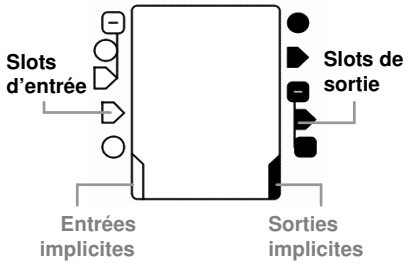
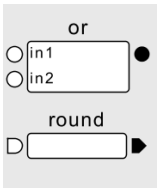


FIG. 3.10 – Un dispositif, avec ses slots d'entrées (à gauche, en blanc) et ses slots de sortie (à droite, en noir).

en sortie en fonction des valeurs qu'il reçoit en entrée. Il peut également déclarer un ensemble de *paramètres* permettant de spécialiser son comportement.



	any	double	int	boolean	string object	null
entrée						
sortie						

FIG. 3.11 – À gauche, un *ou* logique comportant deux booléens en entrée et un booléen en sortie, et un dispositif d'arrondi qui transforme un réel (double) en un entier. À droite, les différents types de slots.

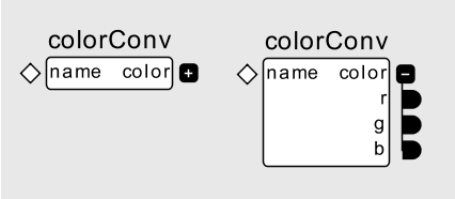


FIG. 3.12 – Un dispositif possédant des slots nommés `name`, `color.r`, `color.g`, et `color.b`. Le slot `color` est un slot composite. À gauche, le dispositif dans sa représentation condensée, à droite dans sa version étendue.

Un **slot** est un état du dispositif qui est accessible aux autres dispositifs. Ces derniers ont la possibilité d'écrire sur le slot s'il s'agit d'un *slot d'entrée*, ou de lire sa valeur s'il s'agit d'un *slot de sortie* (figure 3.10 page précédente). Les slots possèdent des *types* de données simples visibles dans leur représentation graphique (figure 3.11). Par exemple, les slots booléens sont circulaires, les slots entiers ont une forme triangulaire et les types indéfinis sont représentés par un carré. Les slots peuvent être organisés de façon *hiérarchique* par un mécanisme de nommage analogue aux chemins fichiers ou packages Java (voir figure 3.12).

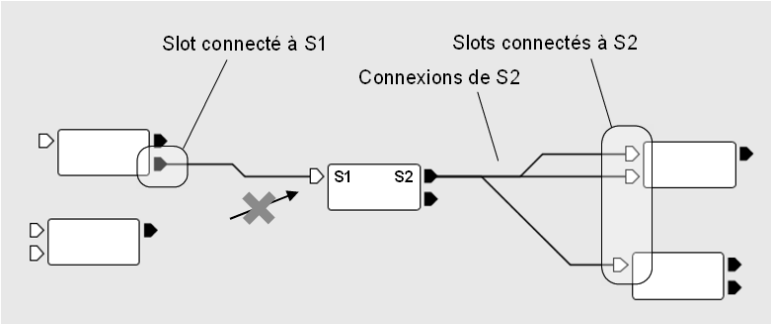


FIG. 3.13 – Exemples de connexions : le slot de sortie *s* comporte trois connexions et le slot d'entrée *e* comporte une connexion. Les slots d'entrée ne peuvent comporter plus d'une connexion.

Une **connexion** est un arc orienté reliant un slot de sortie d'un dispositif à un slot d'entrée d'un autre dispositif (figure 3.13). Deux slots reliés par une connexion partagent la même valeur. Les

connexions multiples sont autorisées sur les slots de sortie mais pas sur les slots d'entrée. En outre, les connexions induisant des dépendances cycliques sont interdites.

3.4.2 Les notions essentielles

Nous présentons ici quatre caractéristiques importantes d'ICOM :

1. Le principe des *entrées/sorties implicites*, qui permet d'évoquer l'asynchronisme, le non-déterminisme et le feedback.
2. La *composition*, qui autorise des configurations à plusieurs niveaux de granularité.
3. Les mécanismes de *mutation*, qui permettent de décrire des dispositifs génériques qui se spécialisent en fonction des connexions et des paramètres.
4. Les *processeurs*, permettant aux dispositifs de décrire plusieurs comportements en exécution possibles.

Les entrées/sorties implicites

Un dispositif opère sur ses slots d'entrée et de sortie mais peut également communiquer avec l'environnement extérieur à la configuration par des entrées ou des sorties *implicites*. La présence d'entrées (resp. de sorties) implicites est graphiquement indiquée par la présence d'une encoche dans le coin inférieur gauche (resp. droit) du dispositif (voir la figure 3.10 page 102).

Les **entrées implicites** traduisent la réception de flux d'informations autres que ceux décrits par les connexions (par exemple, provenant de dispositifs concrets).

Les **sorties implicites** traduisent l'émission d'informations vers d'autres entités que les slots de la configuration d'entrée (par exemple, l'application interactive ou l'écran).

Composition

Notre modèle ne fait pas de distinction entre dispositifs d'entrée, de sortie ou de traitement. Cette unification du concept de dispositifs permet notamment de rendre ces derniers *composables* : par exemple, un dispositif d'entrée associé à un dispositif de sortie est un dispositif comportant à la fois des entrées et des sorties implicites. Deux entités entrent en jeu dans la composition : les slots externes et les dispositifs composites.

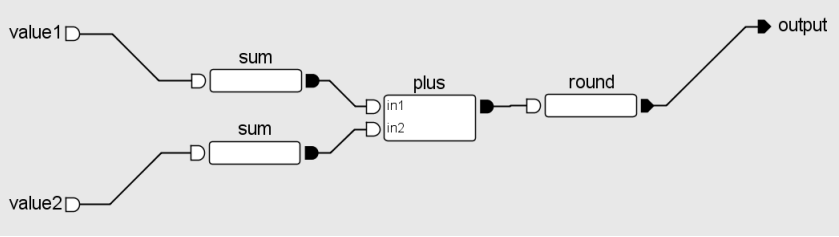


FIG. 3.14 – Une configuration d'entrée comportant trois slots externes.

Une configuration d'entrée peut comporter des **slots externes** (figure 3.14 page ci-contre), qui sont des slots isolés n'appartenant à aucun dispositif. Ces slots définissent en quelque sorte une vue extérieure pour la configuration.

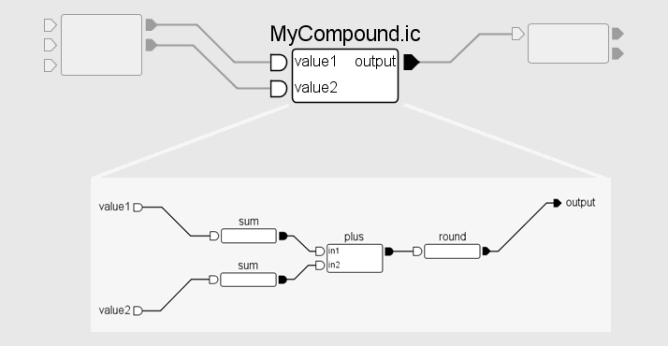


FIG. 3.15 – Un dispositif composite défini par sa configuration fille.

Un **dispositif composite** est un type particulier de dispositif uniquement défini par une *configuration-fille* (figure 3.15). Chacun de ses slots correspond à un slot externe de sa configuration-fille.

L'opération consistant à regrouper un ensemble de dispositifs dans un dispositif composite (composition) et l'opération inverse (décomposition) sont décrits en annexe, section A.4 page 204. La *décomposition totale*, à laquelle nous faisons référence plus loin, consiste à appliquer des décompositions successives jusqu'à obtenir une configuration plate.

Dispositifs mutables

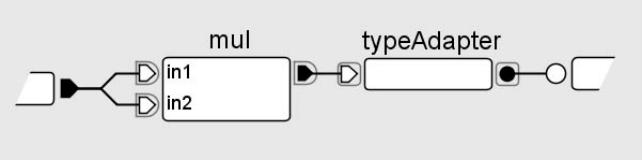


FIG. 3.16 – Deux dispositifs mutables reliés en série. Le premier opère sur des réels ou sur des entiers en fonction de ses connexions en entrée. Le second adapte son type à la fois en entrée et en sortie.

Certains dispositifs génériques sont capables de se spécialiser en fonction des valeurs prises par leurs *paramètres* et des *types* des slots qui leur sont connectés. Cette opération repose sur un mécanisme très général nommé *mutation*, décrit en annexe, section B.3 page 215. Une mutation consiste en une modification du type d'un ou plusieurs slots mutables du dispositif. Un *slot mutable* possède un type concret et un *super-type* décrivant les types concrets qu'il est susceptible d'avoir. Dans la représentation graphique d'un slot mutable, le super-type entoure le type concret (figure 3.16). Certains slots sont en outre *dynamiques*, c'est-à-dire que leur présence dépend du paramétrage du dispositif.

Les processeurs

Un *processeur* est un objet qui encapsule le comportement en exécution d'un dispositif. Plusieurs types de processeurs peuvent être associés à un seul dispositif : un additionneur peut par exemple définir un processeur pour l'addition d'entiers, un pour l'addition de réels et un autre pour la concaténation de chaînes de caractères. Le processeur effectivement employé par l'additionneur au moment de l'exécution dépendra du type concret de son slot de sortie.

L'unique responsabilité d'un processeur est de mettre à jour à la demande (c'est-à-dire à chaque fois qu'il est *déclenché*) les valeurs de ses slots de sortie en fonction des valeurs de ses slots d'entrée (comme évoqué précédemment, un processeur peut également communiquer avec l'environnement).

3.4.3 Aperçu du modèle d'exécution

Lancement

La phase de *lancement* est destinée à préparer la configuration à l'exécution. Elle comporte plusieurs étapes. Pour commencer, l'exécution travaille après *décomposition totale* de la configuration d'entrée. Ensuite, tous les dispositifs de la configuration sont *ouverts* pour leur permettre de s'initialiser. Chaque dispositif retourne lorsqu'il est ouvert un processeur en fonction de sa paramétrisation (valeur de ses paramètres et types concrets de ses slots mutables), ou le *processeur nul* lors d'un échec d'ouverture.

Après avoir récupéré les processeurs, le système crée un objet **Valeur** pour chaque slot de *sortie* de la configuration, objet qui encapsule une valeur concrète (entier, booléen,...) et la partage avec les slots d'entrée connectés. Ces partages de référence autorisent en quelque sorte une propagation instantanée des valeurs à travers les connexions. Pour finir, les processeurs sont *triés topologiquement* en fonction du graphe de dépendances.

L'algorithme du tick

L'algorithme d'exécution se décompose en pas atomiques ou *ticks*. Durant un tick, les processeurs sont parcourus dans l'ordre topologique et chacun d'entre eux est déclenché s'il est de type *actif* (c'est-à-dire qu'il correspond à un dispositif à entrées implicites), ou si au moins une de ses Valeurs en entrée comporte un signal. Le **signal** est un drapeau booléen dans l'objet Valeur qui est toujours à *faux* au début du tick. Lorsqu'un processeur déclenché met explicitement à jour une Valeur (la nouvelle valeur concrète peut être la même), ce drapeau passe à *vrai* pour que les processeurs qui en dépendent puissent être déclenchés.

Un tick consiste donc globalement à déclencher les processeurs actifs (pour la plupart des dispositifs d'entrée) puis à transmettre les mises à jour au reste de la configuration. L'algorithme d'exécution réactif (décrit en annexe, section C.3 page 227) assure que tout changement est correctement propagé et que chaque processeur est activé seulement lorsque c'est nécessaire, *au plus une fois* dans le tick. Les processeurs étant triés topologiquement, la mise à jour de la configuration est effectuée en une seule passe. Cet algorithme est simple et extrêmement efficace.

3.5 Conclusion

Nous avons décrit et motivé dans ce chapitre les principes généraux d'un modèle d'entrée basé sur le paradigme *cascades de dispositifs*. Ce modèle emploie des métaphores simples comme les connexions et les adaptateurs, et il est facile à appréhender aussi bien pour le concepteur que pour l'utilisateur.

Nous avons ensuite concrétisé ce paradigme avec le modèle des *configurations d'entrées*. Une configuration d'entrée est un ensemble de dispositifs système (modules qui décrivent pour la plupart des dispositifs d'entrée physiques) et de dispositifs d'application (modules qui décrivent les entités contrôlables de l'application) reliés par des dispositifs utilitaires (modules qui peuvent être vus comme des adaptateurs ou des techniques d'interaction). Notre modèle repose sur une architecture à flot de données claire et fonctionnelle (qui permet notamment de décrire des modules composables et capables de se spécialiser dynamiquement), et sur un modèle d'exécution réactif qui garantit une réaction immédiate aux actions de l'utilisateur.

Dans les chapitres suivants, nous montrerons par des exemples comment ce modèle permet de décrire la majeure partie des mécanismes en jeu dans la gestion des dispositifs d'entrée dans les systèmes interactifs existants, tout en améliorant la visibilité et la modularité de ces mécanismes. Nous montrerons également comment il peut servir à décrire de nouvelles techniques d'interaction. Le prochain chapitre décrit la boîte à outils en entrée ICON (Input Configurator), développée à partir des principes que nous avons évoqués.

Chapitre 4

La boîte à outils d'entrées ICON

Sommaire

4.1 Introduction

Nous avons développé à partir du modèle des configurations d'entrée décrit dans le chapitre précédent une boîte à outils *« d'entrée »*, dans le but de *valider* et d'*affiner* ce modèle dans un contexte de développement réel, qui comprend un langage de programmation et un système interactif concret (système d'exploitation et boîte à outils graphique).

Nous présentons dans ce chapitre la boîte à outils d'entrée ICON (Input Configurator), dont l'objectif est de :

- Proposer une implémentation concrète du modèle des configurations d'entrée et du moteur réactif nécessaire à leur exécution,
- Offrir une bibliothèque extensible de dispositifs système et utilitaires prêts à l'emploi permettant de construire et tester des exemples de configurations d'entrée,
- Fournir des interfaces et des librairies permettant de développer rapidement des dispositifs d'application et de nouveaux dispositifs système et utilitaires dans un paradigme de programmation orientée objet, ainsi que des services comme la sérialisation (persistance) des configurations d'entrée,
- Procurer aux développeurs et aux utilisateurs moins experts un éditeur interactif qui exploite le modèle visuel des configurations d'entrée et l'instrumente par des techniques d'interaction appropriées.

Nous avons choisi d'implémenter cette boîte à outils avec le langage *Java*, principalement pour sa portabilité et la notoriété dont il bénéficie auprès des programmeurs d'applications. Nous avons également choisi de rendre *Swing* contrôlable par des dispositifs d'application génériques que nous nommerons *dispositifs de boîte à outils graphique*.

Nous décrivons pour commencer des exemples de dispositifs système, utilitaires, de boîte à outils graphiques et d'application, et affinons cette taxonomie dans le contexte d'ICON. Nous décrivons dans un deuxième temps trois notions propres à l'implémentation ICON, à savoir l'interface et les classes abstraites de dispositifs, le mécanisme d'initialisation d'une configuration avec passage du contexte, et le principe des descripteurs employé pour la sérialisation. Nous présentons ensuite l'éditeur interactif de configurations d'entrée, et évoquons par des exemples les différents types de techniques d'interaction que nous avons pu décrire. Enfin, nous donnons avant de conclure des informations sur la distribution, les contributeurs, et les projets utilisant ICON.

4.2 Les dispositifs d'ICON

Comme le modèle des configurations d'entrée, ICON s'appuie sur une classification des dispositifs en *dispositifs système*, *dispositifs utilitaires* et *dispositifs d'application*, à laquelle il ajoute un type particulier de dispositif d'application, les *dispositifs de boîte à outils graphiques*. La *bibliothèque de dispositifs* d'ICON comprend tous les types de dispositifs hormis les dispositifs d'application, qui sont sont décrits par chaque application interactive.

4.2.1 Les dispositifs système

Un *dispositif système* représente une ressource globale, en général un *dispositif d'entrée physique*. Il est :

Partagé : Plusieurs copies d'un dispositif système, si elles sont paramétrées de la même manière, émettront les mêmes données.

Volatile : à un instant donné, un dispositif système peut être présent ou non dans la bibliothèque d'ICON, selon la configuration matérielle et logicielle du système hôte. Lorsqu'une configuration dans laquelle il est utilisé est en cours d'exécution, sa durée de vie est au moins celle de la configuration. Il peut cependant disparaître entre deux exécutions.

L'accès bas-niveau aux dispositifs

ICON repose sur un ensemble d'*APIs d'entrée* existantes afin d'accéder aux dispositifs d'entrée. Chacune de ces APIs d'entrée fournit une interface unifiée pour l'accès aux informations et aux données produites par une classe spécifique de dispositifs, dans un système d'exploitation donné.

ICON produit à son tour une vision unifiée de ces classes de dispositifs. Chaque API d'entrée gérée par ICON génère un ensemble de *dispositifs système* dans la bibliothèque de dispositifs. L'ensemble des dispositifs système est variable, et est fonction de la configuration logicielle (pilotes installés) et physique (dispositifs connectés) du poste de travail.

La plupart des APIs d'entrée fournissent plusieurs niveaux d'accès aux dispositifs, à travers des mécanismes spécifiques de traitement de données et d'abstraction. Dans notre optique, le court-circuitage de ces services par l'utilisation exclusive des services bas-niveau est nécessaire à une utilisation optimale des capacités inhérentes à chaque dispositif d'entrée.

Bien qu'ICON constitue une sur-couche aux APIs d'entrée, il reste à l'écart de toute abstraction simplificatrice en présentant une vision essentiellement *bas niveau* des dispositifs d'entrée. Le modèle de dispositifs à base de canaux d'ICON autorise et même encourage cette vision.

Les dispositifs système existants

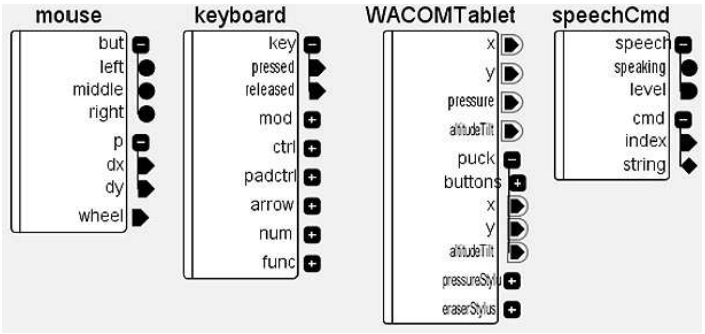


FIG. 4.1 – Exemples de dispositifs système (souris, clavier, tablette graphique et commande vocale).

Actuellement, ICON fournit sous les systèmes Microsoft Windows l'accès à la *souris* (bas-niveau) et au *clavier* (niveau boîte à outils), aux *tablettes graphiques* à travers l'API Wintab [POYNER 96], aux *dispositifs de jeu* et aux *contrôleurs isométriques 3D* à travers l'API DirectInput [MICROSOFT 03A], et à la *reconnaissance vocale* à travers l'API JavaSpeech [SUN 98] (la *reconnaissance gestuelle* également prise en charge, mais sous forme de dispositif utilitaire). Sous X Windows sont accessibles tous les dispositifs implémentés par la *XInput Extension* [PATRICK & SACHS 94]. Un dispositif système particulier nommé *VirtualUser* permet de rejouer des signaux préalablement entregistrés en sortie de dispositifs système. ICON comprend également des dispositifs système de sortie, permettant ainsi de jouer de la musique sur des cartes son, ou faire de la synthèse vocale. La bibliothèque d'ICON est extensible, et des dispositifs système reposant sur d'autres APIs d'entrée peuvent y être ajoutées.

La figure 4.1 page précédente montre des exemples de dispositifs système tels qu'ils apparaissent dans ICON. Le premier dispositif, à gauche, est une *souris* bas niveau, qui émet des valeurs positionnelles relatives *dx* et *dy*. Contrairement aux coordonnées écran traditionnellement utilisées dans les systèmes à événements, ces valeurs traduisent directement les actions effectuées sur le dispositif physique. Elles peuvent ainsi être interprétées indépendamment de toute notion de pointeur (et ne sont pas, en particulier, contraintes aux bords de l'écran).

Le second dispositif est un *clavier* qui émet les changements d'état de chaque touche individuelle, mais qui est également capable d'émettre des « codes touche ». La *tablette graphique* émet des informations spécifiques à chaque pointeur physique (stylet, gomme, souris,...) présent sur tablette, ainsi que des valeurs synthétiques de position, de pression et d'orientation. Enfin, le dispositif de *commande vocale* émet les chaînes de caractères reconnues ou leur indice, conformément au vocabulaire spécifié dans les paramètres du dispositif.

4.2.2 Les dispositifs utilitaires

Les *dispositifs utilitaires* sont les dispositifs de la bibliothèque d'ICON qui ne sont pas des dispositifs système. Ces dispositifs sont *autonomes* (chaque copie d'un dispositif se comporte indépendamment des autres) et *persistants* (un dispositif utilitaire est toujours présent dans la bibliothèque d'ICON, indépendamment de la configuration logicielle et matérielle du système). Nous décrivons dans cette section les différents types de dispositifs utilitaires.

Les dispositifs de traitement

Un dispositif de traitement effectue essentiellement des transformations de données. Ils n'ont pour la plupart pas d'entrées/sorties implicites et sont par conséquent déterministes. D'autres se basent sur un temps absolu, ce qui les rend non-déterministes (le temps n'est pas explicite dans ICON et il n'est accessible que sous forme d'entrée implicite).

La bibliothèque d'ICON comporte une trentaine de dispositifs de traitement qui comprennent les opérateurs mathématiques, les opérateurs logiques, les dispositifs de traitement de signal, les adaptateurs de type et de domaine, et les dispositifs pour le contrôle conditionnel et le routage.

La figure 4.2 page suivante illustre quelques-uns de ces dispositifs. À gauche, un dispositif d'addition acceptant différents types (mutable) et un dispositif de transformation linéaire paramétrable. Ensuite sont représentés deux opérateurs booléens, un dispositif passe-bas paramétrable, un adaptateur de types (cast) automatique et un échantillonneur temporel, et enfin un « passeur » conditionnel

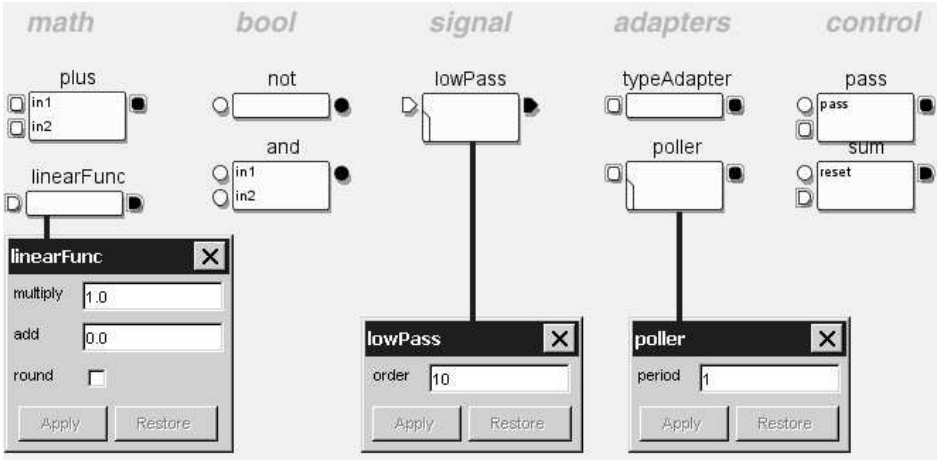


FIG. 4.2 – Exemples de dispositifs de traitement dans ICON, avec les fenêtres de propriétés des dispositifs paramétrables.

et un intégrateur qui permet de transformer des valeurs relatives en valeurs absolues.

La plupart des dispositifs de traitement sont *mutables*, c'est-à-dire qu'ils se spécialisent suivant les types de slots qui leurs sont connectés ou en fonction de leur paramétrage. C'est le cas par exemple du dispositif *plus* de la figure 4.2, qui est capable d'effectuer des additions sur plusieurs types, ou du dispositif *typeAdapter*, qui se spécialise selon les types qui lui sont connectés en entrée et en sortie.

Les dispositifs à retour graphique

Les dispositifs à retour graphique sont des dispositifs utilitaires qui produisent un affichage graphique afin d'émettre des informations vers l'utilisateur (*feedback*). Ces dispositifs comportent des sorties implicites, mais contrairement aux dispositifs système, ils sont persistants car indépendants de toute configuration logicielle. Concernant la configuration matérielle, nous supposons que dans les systèmes ICON un écran est toujours disponible. Dans le cas contraire (prise en compte de systèmes embarqués, par exemple), ces dispositifs appartiendraient à la catégorie système.

Les dispositifs à retour graphique actuellement présents dans ICON sont représentés sur la figure 4.3 page suivante. À gauche apparaissent les dispositifs d'affichage de données brutes, essentiellement utilisés pour le débogage de configurations. Le premier affiche l'historique des données qu'il reçoit sur la sortie standard. Le second affiche ses dernières valeurs reçues dans une fenêtre de dialogue.

Les dispositifs suivants animent des objets graphiques en incrustation (*overlay*) par-dessus les autres éléments de l'écran. Le dispositif curseur permet d'animer un pointeur en fonction des positions en coordonnées écran qu'il reçoit, et comporte également des slots booléens destinés à recevoir et transmettre des états de boutons. Ce dernier aspect lui confère un rôle de dispositif virtuel. La fenêtre de propriétés de ce dispositif, représentée sur la figure, offre le choix entre plusieurs pointeurs prédéfinis. Chaque copie d'un dispositif curseur est autonome (hormis les recouvrements visuels), ce qui autorise les curseurs multiples.

Les deux derniers dispositifs offrent des retours graphiques similaires au curseur, bien que plus

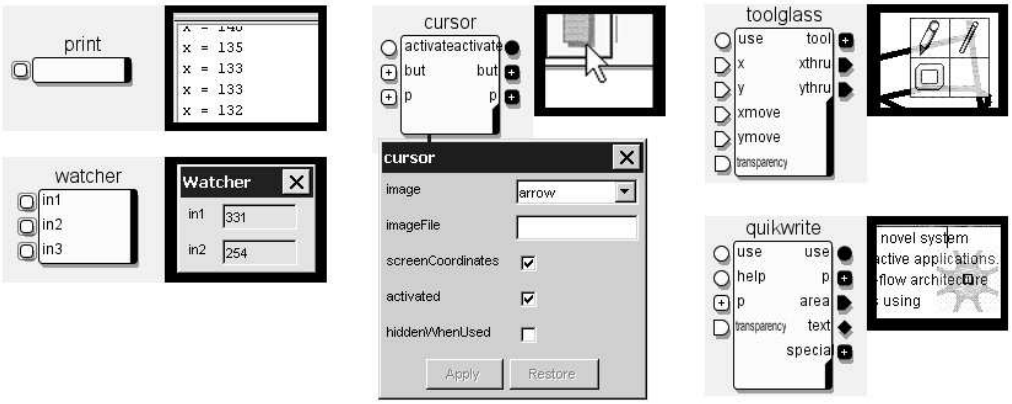


FIG. 4.3 – Dispositifs à retour graphique dans ICON.

avancés : le dispositif *toolglass* permet d'animer une barre d'outils semi-transparente [BIER *et al.* 93], et le dispositif *quikwrite* permet de contrôler un éditeur de texte avec uniquement un pointeur. Ces dispositifs, qui décrivent également des techniques d'interaction, sont décrits plus en détail par la suite, dans la section 4.4.4 page 139.

Les affichages en incrustation sont actuellement implémentés au niveau de la boîte à outils Swing, la plupart des systèmes d'exploitation (dont Microsoft Windows) n'offrant pas ce type de service. Les mécanismes employés actuellement permettent d'afficher et d'animer par-dessus n'importe quelle fenêtre Swing un ou plusieurs composants graphiques semi-transparents, sans modifier le code de l'application. Ces affichages restent cependant limités aux fenêtres Swing, et les dispositifs qui les emploient appartiennent par conséquent à la catégorie des *dispositifs de boîte à outils graphique* (voir section suivante), bien qu'en réalité ils ne communiquent pas avec les composants interactifs. Mais il n'est pas exclu qu'à terme, les plate-formes actuelles fournissent des services d'incrustation et que les dispositifs de retour graphique sophistiqués soient fournis comme des services de base dans ICON.

D'autres dispositifs à retour graphique, basés ou non sur les mécanismes d'incrustation, peuvent être implémentés et ajoutés à la bibliothèque d'ICON.

4.2.3 Les dispositifs de boîte à outils graphique

Les dispositifs de boîte à outils graphique, situés à mi-chemin entre la catégorie des dispositifs utilitaires et celle de dispositifs d'applications, permettent un contrôle générique de ces dernières. À travers des entrées ou des sorties implicites, ces dispositifs communiquent avec les composants interactifs (*widgets*) des fenêtres développées avec la boîte à outils. Les références vers les fenêtres contrôlées sont passées en paramètre à travers le contexte d'exécution (→ section 4.3.2 page 121). ICON comporte des dispositifs de boîte à outils liés à *Swing*. Nous les décrivons ici.

Sélecteurs et manipulateurs

Il existe deux types de dispositifs de boîte à outils graphique :

Les sélecteurs. Les sélecteurs sont des dispositifs qui *émettent en sortie des références vers des*

composants interactifs. Chaque sélecteur décrit une *stratégie de sélection* pour ces composants. Les sélecteurs standards des boîtes à outils sont la sélection positionnelle (*picking*) et le focus. D'autres stratégies peuvent être développées, comme les références statiques ou les sélections multiples.

Les manipulateurs. Les manipulateurs sont des dispositifs qui *prennent en entrée des références vers des composants interactifs*. Chaque manipulateur possède des fonctionnalités de *contrôle* propres à une classe donnée de composants, et emploie les références reçues en entrée pour déterminer sur quelles instances opérer. Parmi ces dispositifs, un **manipulateur de surface** offre un contrôle générique des composants interactifs. Celui-ci expose toutes les opérations possibles sur ces composants, telles que l'émission d'événements positionnels. Ensuite, chaque **manipulateur de modèle** décrit les opérations supplémentaires qu'il est possible d'effectuer sur une classe particulière de composants : modification de la valeur d'une barre de défilement ou insertion de texte dans une zone de texte, par exemple.

Les dispositifs Swing

ICON inclut un ensemble de dispositifs Swing, dont le rôle est de permettre d'interagir, de façon standard ou non, avec toute application reposant sur cette boîte à outils. Afin de permettre de repenser l'intégralité de l'interaction avec Swing et d'éviter les interférences avec ses techniques d'interaction standard, ICON désactive par défaut toute gestion événementielle des entrées lors de l'exécution d'une configuration :

Désactivation du clavier. Lorsque le dispositif clavier est utilisé dans une configuration en cours d'exécution, ICON désactive *l'interprétation des événements clavier* dans Swing en consommant ces événements dans la file d'événements Java. ICON détourne également la combinaison Alt+C, utilisée pour stopper la configuration.

Désactivation de la souris. Lorsque le dispositif souris est utilisé dans une configuration en cours d'exécution, ICON désactive *l'interprétation des événements souris*, ainsi que le *curseur système*. Ce dernier est rendu invisible et maintenu dans une fenêtre Swing.

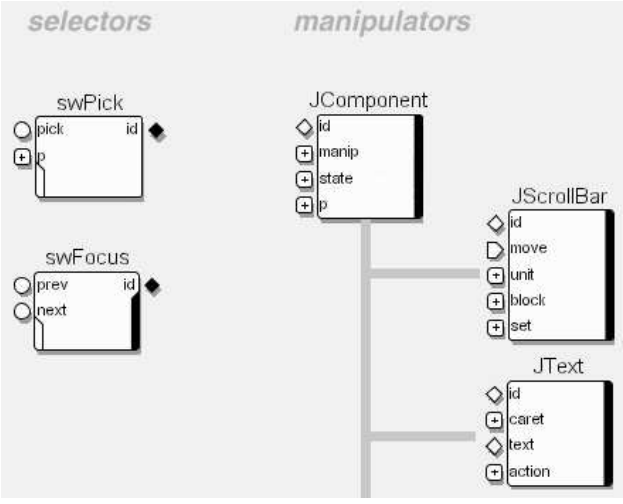


FIG. 4.4 – Les dispositifs de la boîte à outils Swing.

Les dispositifs Swing sont représentés sur la figure 4.4 page précédente. Ceux-ci incluent deux sélecteurs standards : le picking et le focus. Le premier émet une référence vers le composant qui se trouve sous la dernière position reçue en entrée, lorsque son slot *pick* est à *vrai*. Le dispositif de focus communique avec le système de focus de Swing : il notifie des changements de focus, et permet également de contrôler ce focus en passant au composant suivant ou précédent.

Le dispositif suivant est un manipulateur de surface qui déclare toutes les actions qu'il est possible d'effectuer sur un objet de type *JComponent*. Un certain nombre de slots d'entrée permettent de simuler les manipulations positionnelles standard avec une granularité assez fine. Ces slots incluent une position, trois boutons, et des modificateurs (touches de contrôle, double-clic). À partir de ces informations, le dispositif construit des événements souris synthétiques (de type *MouseClicked*, *MouseMoved*, *MouseDragged*, etc.), qui sont émis vers le composant en cours. D'autres slots permettent de modifier l'état interne du composant, comme sa visibilité ou son focus.

Les dispositif *JScrollbar* offre différentes manières de contrôler la valeur courante des barres de défilement et des curseurs (*sliders*). Le dispositif *JText* permet quant à lui de contrôler sur les composants textuels le placement du caret et d'y insérer du texte. Il expose en outre toutes les actions abstraites déclarées par le composant Swing, telles que la suppression de texte ou le copier-coller.

Ces 5 dispositifs constituent une bibliothèque minimale pour expérimenter le contrôle des applications Swing avec ICON (un projet extérieur à cette thèse étant en cours pour développer un bibliothèque complète de dispositifs reposant sur la boîte à outils Jazz/Piccolo). Notre bibliothèque expérimentale offre cependant déjà une grande richesse quant aux possibilités de construction de techniques d'interaction non standards, et nous a permis de valider les principes de base des dispositifs de boîtes à outils, à savoir les notions de sélecteurs et de manipulateurs.

Notons enfin que les dispositifs Swing maintiennent une référence vers une fenêtre unique et opèrent sur un seul composant de cette fenêtre à la fois, mais qu'il est tout à fait envisageable d'étendre ces dispositifs pour gérer des listes de références, aussi bien pour les fenêtres que pour les composants.

Les systèmes de coordonnées dans ICON

ICON laisse les détails de gestion des systèmes de coordonnées aux boîtes à outils graphiques et aux applications. Il emploie un système de coordonnées unique, à savoir des *coordonnées écran en double précision*. Il s'agit d'une convention que les dispositifs d'entrée positionnels, les dispositifs de boîtes à outils graphiques et les applications partagent.

L'usage de la résolution de l'écran comme système de référence évite les conversions multiples et permet d'attribuer une sémantique précise aux slots positionnels présents en entrée ou en sortie sur des dispositifs comme le *Pick*. L'emploi de valeurs réelles plutôt qu'entières possède quant à lui plusieurs avantages. Tout d'abord, il réduit de façon importante les pertes de précision dans les conversions de coordonnées locales en coordonnées écran. Ensuite, certains dispositifs positionnels absolus comme la tablette graphique possèdent une résolution bien plus importante que celle de l'écran[1], et cette résolution peut ainsi être exploitée par les applications (par exemple les applications de dessin) et par certaines techniques d'interaction comme la reconnaissance gestuelle. Par ailleurs, certaines boîtes

[1]Le dispositif tablette produit par défaut des positions dans le système de coordonnées écran de référence, mais est également capable de fournir au besoin des positions entières dans son propre système de coordonnées. Il est plus généralement souhaitable, pour rester au plus près du dispositif physique, que l'attribution d'une sémantique positionnelle à certains canaux de dispositifs d'entrée ne soit pas systématique.

à outils possèdent un mode d'affichage graphique supérieur à la résolution de l'écran (nommé anti-aliasing dans Swing), qui peut être avantageusement exploité par les dispositifs de feedback ou les applications.

4.2.4 Les dispositifs d'application

Bien qu'une application interactive puisse être contrôlée de manière générique par les dispositifs de boîte à outils graphique, les développeurs peuvent étendre la contrôlabilité de leur application en implémentant et en déclarant des dispositifs spécifiques : ce sont les dispositifs d'application.

Les *dispositifs d'application* sont les dispositifs qui n'appartiennent pas à la bibliothèque d'ICON et qui sont fournis par les applications.

L'application ICONDraw

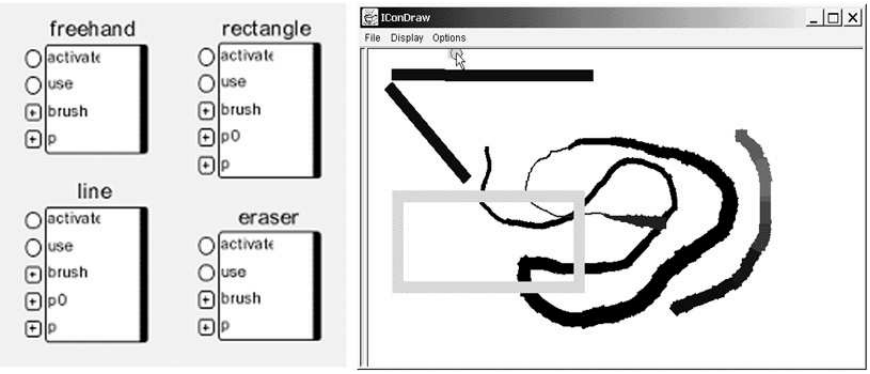


FIG. 4.5 – Les dispositifs de l'application de dessin ICONDraw.

La figure 4.5 montre une application de dessin en Java, ICONDraw (à droite), qui a été écrite pour être entièrement reconfigurable par ICON. Cette application est totalement isolée des événements Java standards, hormis les composants interactifs Swing situés autour de la zone de dessin. Elle décrit simplement la façon dont elle s'attend à être contrôlée en exposant ses outils de dessin sous forme de dispositifs d'application (figure 4.5, image de gauche).

Les outils de dessin sont au nombre de quatre : dessin à main levée, tracé de lignes et de rectangles, et gomme. Ces outils sont contrôlés par l'émission, à travers les slots *p0* et *p*, d'une ou deux séries de positions (les lignes et les rectangles sont définies par leurs deux extrémités). Un outil peut également être activé ou désactivé (slot *activate*) et recevoir des attributs graphiques tels que la taille de la brosse et les composantes couleur RGBA (slot *brush*).

Les outils de dessin gèrent également un retour graphique, ainsi que deux modes d'utilisation, le mode courant étant spécifié par le slot *use*. Dans le mode « non utilisé », un retour graphique est effectué sur la position et les attributs de la brosse courante. En mode « utilisé », le retour graphique est effectué sur la forme géométrique en cours de création. Cette forme est validée et ajoutée au dessin lors du passage suivant au mode « non utilisé ».

Types de dispositifs d'application

Nous dégageons trois grands types de dispositifs d'application, adaptés à un contexte de programmation orientée-objet comme java :

Les dispositifs de classe. Un dispositif de classe est un dispositif d'application qui représente une classe d'objets. Les slots décrivent les opérations qui peuvent être effectuées par l'utilisateur sur les objets de cette classe. Les dispositifs de classe comportent en entrée un slot de type objet, qui spécifie la ou les instances cibles. Les *JComponent* et leurs dérivés, décrits précédemment dans la section 4.2.3 page 114, en sont des exemples. Les dispositifs de classe sont en général utiles pour contrôler des objets créés et supprimés dynamiquement pendant l'exécution d'une configuration.

Les dispositifs d'instance statiques. Les dispositifs d'instance statiques représentent des instances particulières de classes. Seules les instances créées puis convenablement déclarées avant l'initialisation d'une configuration d'entrée seront visibles et contrôlables. Par exemple, après la création et l'initialisation d'une fenêtre, un dispositif d'instance statique pourra être créé pour chaque composant interactif présent dans la fenêtre. Les dispositifs d'instance statiques sont utiles pour le contrôle dédié.

Les dispositifs d'instance dynamiques. Les dispositifs d'instance dynamiques représentent des instances de classes qui sont créées à la demande d'ICON. Lors du lancement d'une configuration d'entrée, chaque dispositif d'instance dynamique présent dans la configuration est ouvert par un appel à sa méthode *open*. Dans cette méthode, un gestionnaire d'instances de l'application crée l'instance et l'*installe*, c'est-à-dire la lie au reste de l'application. D'autres objets peuvent être créés durant cette procédure d'installation. Les dispositifs de dessin d'ICONDraw sont des dispositifs d'instance dynamiques : un outil de dessin est instancié pour chaque dispositif présent dans la configuration. Plusieurs instances du même outil de dessin peuvent coexister, et il n'existe pas de limite quant au nombre d'outils présents dans la fenêtre de dessin. Les dispositifs d'instance dynamiques, très puissants, sont utiles pour décrire des *outils*, c'est-à-dire des objets de manipulation qui ne font pas partie intégrante du modèle interne de l'application.

4.3 Programmation avec ICON

Pour programmer avec la boîte à outils ICON, il n'est pas nécessaire de connaître la structure et les mécanismes de base (typage et exécution) des configurations d'entrée, décrits indépendamment par le modèle ICOM.

Cependant, l'extension de la bibliothèque d'ICON, le développement d'applications interactives configurables et la distribution de configurations d'entrées réutilisables nécessitent la connaissance de quelques mécanismes essentiels propres à la boîte à outils ICON. Nous les décrivons dans cette partie.

4.3.1 Implémentation de nouveaux dispositifs

Dispositifs programmables

ICON fournit un dispositif utilitaire *Script* qui permet de décrire des comportements simples en JavaScript, directement dans l'éditeur de configurations et sans nécessité de recompilation. La liste des

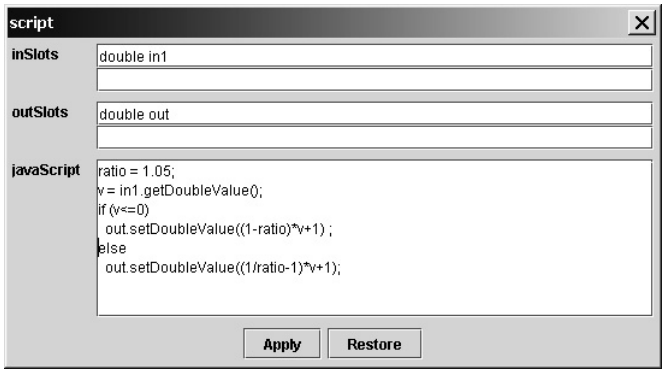


FIG. 4.6 – Fenêtre de propriétés du dispositif *Script*.

slots du dispositif ainsi que le code de traitement sont saisis dans la fenêtre de propriétés du dispositif, illustrée figure 4.6. Cet exemple décrit une fonction de transfert utilisée dans le contrôle d'un zoom par un dispositif isométrique.

L'interface Device

Comme nous le verrons dans cette section, l'implémentation en Java d'un dispositif simple est relativement élémentaire. Les fonctionnalités d'un dispositif d'entrée sont décrites par l'interface *Device* dans l'API ICON. Cette interface est reproduite, avec les commentaires, sur la figure 4.7 page suivante. La partie exécution des dispositifs d'entrée est décrite par l'interface *Processor*, reproduite sur la même figure.

En pratique, un dispositif dérive toujours de la classe *AbstractDevice*. Cette classe implémente un certain nombre de comportements par défaut communs à une grande majorité des dispositifs, à savoir, la gestion des slots d'entrée et de sortie à travers les méthodes *addIn* et *addOut*, l'auto-duplication (consistant en une ré-instanciation de la classe et la copie des valeurs des paramètres), et la gestion d'un processeur unique.

Exemple : le dispositif DColorConv

La figure 4.8 page 121 montre un exemple de dispositif utilitaire qui a été ajouté à la bibliothèque d'ICON pour permettre un contrôle vocal des attributs de la brosse dans ICONDraw. Ce dispositif *DColorConv* convertit des noms de couleur en leurs composantes RGB.

Écrire un dispositif utilitaire consiste essentiellement à déclarer ses slots d'entrée et de sortie, et à remplir le corps de sa méthode de mise à jour *update*, pour assigner les valeurs des slots de sortie en fonction de celles des slots d'entrée (la gestion des tables de couleurs n'est pas montrée dans le code). À chaque fois qu'une des valeurs en entrée changera, cette méthode de mise à jour sera automatiquement appelée par la machine réactive d'ICON.

La classe *AbstractDevice* gère un processeur unique : elle implémente l'interface *Processor*, et sa méthode *open()* retourne *this* par défaut. La gestion de processeurs multiples, utile pour faire de la spécialisation de code, s'effectue par dérivation de la méthode *open()* et l'utilisation d'*inner classes*.

```
public interface Device {

  public static String[] AUTO_PROPERTIES =
    new String[] {"Auto"};

  /**
   * Retourne le nom du dispositif.
   */
  public String getName();

  /**
   * Retourne des informations sur le
   * dispositif à destination de
   * l'utilisateur, ou null.
   */
  public DeviceInfo getInfo();

  /**
   * Retourne les slots d'entrée du dispositif.
   */
  public In[] getIns();

  /**
   * Retourne les slots de sortie du
   * dispositif.
   */
  public Out[] getOuts();

  /**
   * Spécifie si ce dispositif comporte des
   * entrées implicites.
   */
  public boolean hasExternalInput();

  /**
   * Spécifie si ce dispositif comporte des
   * sorties implicites.
   */
  public boolean hasExternalOutput();

  /**
   * Retourne les paramètres du dispositif.
   * Pour chaque paramètre X, la classe doit
   * déclarer deux accesseurs setX et getX.
   * En retournant AUTO_PROPERTIES, les
   * paramètres sont automatiquement déduits
   * des accesseurs de la classe.
   */
  public String[] getProperties();

  /**
   * Spécifie si ce dispositif est duplicable
   */
  public boolean isCopiable();

  /**
   * Retourne un dispositif avec les mêmes
   * fonctionnalités, ou null si le dispositif
   * n'est pas duplicable (instance statique).
   */
  public Device copy();

  /**
   * Spécifie ci ce dispositif peut
   * actuellement être ouvert (i.e.
   * s'il est correctement paramétré).
   */
  public boolean isOpenable();

  /**
   * Ouvre le dispositif pour le préparer à
   * l'exécution. Toute allocation de
   * ressources doit s'effectuer ici.
   * Le contexte d'exécution permet, le cas
   * échéant, de passer au dispositif les
   * références nécessaires à son exécution
   * (fenêtres, etc.)
   * Retourne un Processor si l'ouverture a
   * réussi, null dans le cas contraire (voir
   * l'interface Processor, plus bas).
   */
  public Processor open(OpenContext context);

  /**
   * Cette méthode est appelée lorsque le
   * dispositif n'est plus utilisé. Les
   * ressources doivent être désallouées ici.
   */
  public void close();

  /**
   * Retourne un court message d'erreur
   * explicatif, si isOpenable() retourne faux
   * ou après un échec d'ouverture.
   */
  public String getError();

  /**
   * Active ou désactive le dispositif.
   */
  public void setEnabled(boolean enabled);

}
```

```
public interface Processor {
  /**
   * Cette méthode est appelée lors du premier
   * tick d'exécution.
   */
  void init();

  /**
   * Cette méthode est appelée lorsqu'un signal
   * est présent dans l'un des slots d'entrée.
   */
  void update();
}
```

FIG. 4.7 – Les interfaces *Device* et *Processor*.

```
public class DColorConv extends AbstractDevice {

   // déclaration des slots

   public final In name = addIn("name", SlotType.STRING);
   public final Out r = addOut("color.r", SlotType.DOUBLE);
   public final Out g = addOut("color.g", SlotType.DOUBLE);
   public final Out b = addOut("color.b", SlotType.DOUBLE);


(...)

   // mise à jour des slots de sortie

   public void update() {
      if (name.hasSignal()) {
         String s = name.getStringValue();
         boolean found = false;
         for (int i=0; i<COLOR_NAMES.length && !found; i++)
            found = (COLOR_NAMES[i].equals(s));
         if (found) {
            r.setDoubleValue(COLOR_R[i-1]);
            g.setDoubleValue(COLOR_G[i-1]);
            b.setDoubleValue(COLOR_B[i-1]);
         }
      }
   }

}
```

FIG. 4.8 – Code java du dispositif *DColorConv*.

Le dispositif peut être rendu paramétrable (par exemple, une liste de couleurs personnalisable), simplement en ajoutant des méthodes de type accesseurs à la classe. ICON déduit automatiquement les noms de paramètres par introspection et génère la fenêtre de propriétés correspondante dans l'éditeur.

L'implémentation d'un dispositif d'application est similaire à celle d'un dispositif utilitaire comme *DColorConv*, à ceci près que les assignations de slots de sortie sont remplacées par des appels de méthodes dans l'application elle-même.

4.3.2 Initialisation d'une configuration

Principe général

Pour être configurable par ICON, une application doit créer une configuration d'entrée et faire le lien entre cette configuration et ses propres objets. L'ensemble des mécanismes de communication entre l'application et sa configuration d'entrée est résumé sur la figure 4.9. La création d'une configuration d'entrée nécessite la déclaration d'un *dossier de dispositifs* et le passage d'un *contexte d'exécution*.

Le *dossier de dispositifs* contient l'ensemble des dispositifs qui pourront être utilisés dans la configuration, et qui seront visibles dans la partie gauche de l'éditeur de configurations (voir section 4.4.1 page 127). En général, il s'agit de la bibliothèque standard d'ICON comportant tous les dispositifs système et utilitaires organisés en sous-dossiers, à laquelle est ajouté un dossier spécifique à l'application. Comme tout dossier, ce dernier est un conteneur de *dispositifs-prototypes*, instances de dispositifs qui sont dupliquées pour être ajoutées à la configuration. Un ou plusieurs prototypes peuvent

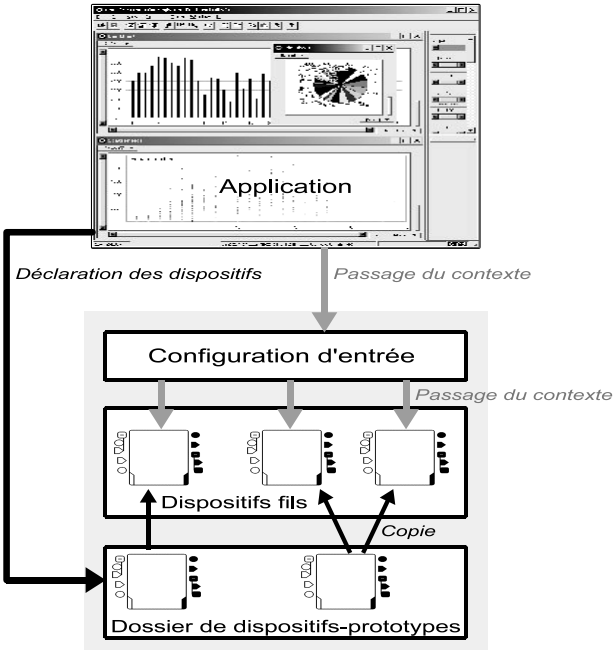


FIG. 4.9 – Initialisation d'une configuration d'entrée dans une application interactive.

être déclarés par classe de dispositif, ce qui permet notamment de fournir des ensembles de dispositifs pré-paramétrés, ou de décrire des dispositifs d'instance statiques.

Le *contexte d'exécution* est une table de hachage qui contient toutes les informations nécessaires à l'exécution de l'ensemble des dispositifs du dossier. Les dispositifs de boîte à outils graphique, que nous avons évoqués précédemment, nécessitent une référence vers la ou les fenêtres qu'ils contrôlent. En outre, l'application possède ses propres dispositifs qui requièrent des références vers des objets de celle-ci, références qui sont ajoutées au contexte sur des clés spécifiques. Lors du lancement de la configuration d'entrée, chaque dispositif reçoit dans sa méthode *open* le contexte d'exécution d'où il va extraire les informations pertinentes, et le cas échéant produire un échec d'ouverture si celles-ci sont indisponibles.

Implémentation d'une application configurable

Le code type d'une application configurable par ICON est donné sur la figure 4.10 page ci-contre. Ce code présente trois parties principales :

A. Initialisation de la configuration d'entrée. Ce code est ajouté à la fin de la méthode constructeur de l'application. Le dossier de dispositifs est créée en passant le dossier de l'application (*MyDevices*) au constructeur de la bibliothèque standard d'ICON (*FRoot*). La configuration d'entrée (ainsi que sa fenêtre d'édition) est ensuite créée, puis exécutée.

B. Gestion de la configuration. Une gestion minimale de la configuration consiste en l'ajout d'un élément interactif (élément de menu, par exemple) qui déclenche le passage en mode édition (méthode *editInput()*), et en la gestion propre de la désallocation des dispositifs lors de la fermeture

```
public class MyApp extends JFrame {

      static final File default_ic = new File("IC files/default.ic");
      final Configuration ic;
      final JInputEditor editor;

      (...)

      public MyApp() {

            (...)

A           // 1. Création de la bibliothèque de dispositifs
            DeviceFolder devices = new FRoot(new MyDevices());
            // 2. Création du contexte d'exécution
            OpenContext context = new OpenContext();
            context.setValue(OpenContext.KEY_JFRAME, this);
            context.setValue("MY_APP", this);
            // 3. Création de la configuration d'entrée
            ic = new Configuration(context, devices);
            // 4. Chargement de la configuration par défaut
            ic.load(default_ic);
            // 5. Création d'une fenêtre ICon
            editor = new JInputEditor(ic);
            // 6. Ouverture et démarrage de la configuration
            ic.open();
            ic.start();
      }

      (...)

B     // Appelé par un bouton ou un élément de menu
      public void editInput() {
            // Arrêt et édition de la configuration
            ic.stop();
            editor.setVisible(true);
      }

      // Appelé par un bouton ou un élément de menu
      public void quit() {
            // Arrêt et fermeture de la configuration
            ic.stop();
            ic.close();
            System.exit(0);
      }
}

// Liste des dispositifs de cette application                    C
class MyDevices extends AbstractFolder {
      public MyDevices() {
            super("MyApp");
            // Création et ajout des prototypes de dispositifs
            add(new AppDevice1());
            add(new AppDevice2());
            add(new AppDevice3());
      }
}
```

FIG. 4.10 – Code type d'une application configurable en entrée. Les dispositifs d'application ne sont pas représentés.

de l'application (méthode *quit()*).

C. Dossier de dispositifs. Les dispositifs-prototypes de l'application sont déclarés par dérivation d'une classe abstraite (*AbstractFolder*).

4.3.3 Sérialisation et descripteurs

Pour pouvoir être sauvegardées et chargées, les configurations d'entrée doivent être *sérialisables*. La sérialisation consiste à convertir un objet en une série d'octets (que nous nommerons *signature*) pour pouvoir, lors d'une exécution ultérieure, recréer le même objet par le processus inverse. Dans cette section, nous évoquerons essentiellement la sérialisation de dispositifs.

Les difficultés de la sérialisation de dispositifs

Établir la signature d'un dispositif système n'est pas trivial. En voici quelques raisons :

1. Un dispositif système n'est pas entièrement décrit par sa classe. Par exemple, deux dispositifs physiques de même type peuvent être présents, et produire dans ICON deux dispositifs qui dérivent de la même classe.
2. Toutes les APIs d'entrée ne garantissent pas un mécanisme déterministe de détection et d'identification de dispositifs. L'ordre dans lequel les dispositifs sont listés peut également changer.
3. Entre deux exécutions, un dispositif d'entrée physique peut avoir été débranché et rebranché sur un autre connecteur : il devra tout de même être retrouvé, puisqu'il s'agit du même dispositif.
4. Deux dispositifs d'entrée physiques identiques (même marque, même modèle) peuvent être utilisés conjointement dans une configuration et y jouer un rôle différent. Il va de soi que ces dispositifs sont physiquement placés différemment sur l'espace de travail (main droite et main gauche, par exemple). D'une exécution à l'autre, leurs rôles respectifs ne devra donc pas être interverti.

Établir des signatures les plus précises possibles, notamment en s'appuyant sur des identifiants uniques (à condition qu'ils soient fournis par l'API d'entrée), peut résoudre en partie ces problèmes. Des interrogations subsistent cependant : faut-il prendre en compte l'identifiant matériel du dispositif ou celui du connecteur sur lequel il est branché ? Si seul l'identifiant matériel est utilisé, la situation 4. évoquée précédemment ne sera pas résolue. Dans le cas contraire, c'est la situation 3. qui posera problème.

D'autres arguments vont à l'encontre de l'emploi d'identifiants uniques. Le remplacement d'une souris par une autre, par exemple, ne devrait pas rendre une configuration standard inutilisable. De même, certaines configurations d'entrée devraient pouvoir être réutilisables d'un poste à l'autre et d'un utilisateur à l'autre. Il est clair que la réutilisabilité des configurations d'entrée et l'identification précise des dispositifs physiques pour la persistance sont deux exigences incompatibles, et c'est pourquoi il est essentiel d'employer des stratégies de signature adaptées à chaque contexte.

Notons enfin que ces problèmes sont généralisables à tous les dispositifs comportant des entrées ou sorties implicites (les dispositifs d'application en particulier), car, d'une exécution à l'autre, il faut être capable de reconstruire ou de retrouver les objets extérieurs avec lesquels ils communiquaient.

La gestion de la sérialisation dans ICON est d'une grande souplesse, grâce au principe de *descripteurs* sur lequel il repose. La signature d'un dispositif est obtenue par la signature d'un de ses descripteurs. Des descripteurs plus ou moins sélectifs peuvent être construits, chaque dossier de dispositifs ayant la charge de fournir des descripteurs stricts (uniques) pour ses dispositifs fils.

Les descripteurs : définition

Un descripteur est un *ensemble non nécessairement dénombrable de dispositifs*. En outre, chaque descripteur est lié à une représentation textuelle, tel qu'il existe une *bijection entre l'ensemble des descripteurs et l'ensemble de leurs représentations textuelles*.

Dans ICON, l'interface `DeviceDescriptor` est définie par les méthodes suivantes :

```
public boolean contains(Device d)
public String getString()
```

Une *classe de descripteurs* est une classe :

1. Qui implémente l'interface `DeviceDescriptor`
2. Dont le nom est préfixé par `DD`
3. Qui comporte un constructeur du type `public DDFoo(String s)`
4. Qui vérifie pour toute instance `desc` :
   `desc` $\equiv$ `new DDFoo(desc.getString())`
   (équivalence de leurs fonctions `contains` et `getString`).

Un *descripteur* est une instance d'une classe de descripteurs. La *signature* d'un descripteur est obtenue par concaténation de son nom de classe sans le préfixe `DD`, du signe `=`, et de sa représentation textuelle.

Exemple : pour une instance de `DDFoo` dont `getString()=4`, la signature est `foo=4`.

Les classes de descripteurs d'ICON

Il existe plusieurs classes de descripteurs prédéfinies dans ICON. Il est ainsi possible de décrire des ensembles de dispositifs possédant un *nom* donné, dérivant d'une *classe* donnée, possédant un ensemble minimal de *slots* (noms et/ou types), ou provenant d'un *dossier* donné. Ces descripteurs sont extrêmement puissants et la plupart permettent d'utiliser des caractères génériques (`*` et `?`). De nouvelles classes de descripteurs peuvent créées et employées.

Les descripteurs sont composables : les *meta-descripteurs* sont des descripteurs qui comportent un ou plusieurs descripteurs-fils. La plupart permettent d'effectuer des opérations ensemblistes. Le descripteur `DDAnd`, par exemple, effectue des intersections de descripteurs. À titre d'exemple, la phrase « *tous les dispositifs dont le nom commence par "mouse" et qui dérivent d'une classe "DMouse"* » sera construite ainsi :

```
DeviceDescriptor mydesc =
new DDAnd(new DDName("mouse*"), new DDClass("*.DMouse"));
```

ou pourra être spécifiée par la signature suivante :

```
DeviceDescriptor desc = DescriptorUtilities.createDD("
and {
name=mouse*;
class=*.DMouse;
}
");
```

L'algorithme de chargement

Lors du chargement d'un fichier de configuration, et pour chaque descripteur de ce fichier, le dossier de dispositifs est parcouru jusqu'à ce qu'un dispositif-prototype convienne (`contains(d)==true`). Celui-ci est alors copié, ajouté à la configuration et n'est plus pris en compte dans les recherches ultérieures.

Le descripteur `DDProperties`, qui décrit des paramétrages pour les dispositifs, joue un rôle particulier dans l'algorithme de désérialisation. Lorsqu'un dispositif-prototype correct a été trouvé, puis copié, l'algorithme lui applique le paramétrage spécifié par le descripteur lorsqu'il existe. Si ce paramétrage provoque une erreur (paramètre inexistant ou valeur incorrecte), l'algorithme reprend la recherche.

Lors d'une désérialisation, plusieurs dispositifs-prototypes peuvent répondre à un descripteur. C'est le premier rencontré qui est choisi, puis éliminé des choix ultérieurs. L'algorithme effectue cependant des retours arrière lorsqu'il aboutit à une impasse (dispositif introuvable).

Les descripteurs stricts

Lors d'une sauvegarde, ICON utilise par défaut des *descripteurs stricts*, garantissant l'unicité. Chaque dossier de dispositifs est chargé de fournir des descripteurs stricts pour ses dispositifs, par la méthode `getDescriptor(Device d)`. Dans la classe `AbstractFolder`, cette méthode retourne simplement un descripteur sur la classe du dispositif passé en paramètre. Ce comportement par défaut convient aux dispositifs utilitaires et à la plupart des dispositifs d'application. Le plus souvent, l'implémentation d'une application configurable par ICON ne nécessite donc aucune connaissance sur les descripteurs.

Pour un dossier comportant plusieurs prototypes de même classe, il devient cependant nécessaire de retourner un descripteur plus précis, portant par exemple sur le nom du dispositif. Les dossiers de dispositifs système, quant à eux, devront retourner une liste complète des caractéristiques physiques du dispositif pour s'assurer que deux dispositifs ne possèdent jamais le même descripteur. Des classes spécifiques de descripteurs pourront également être créées et employées.

Le langage ICSCRIPT

Les fichiers de configurations sont générés dans un langage de script dédié nommé ICSCRIPT. Un exemple de fichier de configuration est donné figure 4.11 page 128. Celui-ci est partagé en trois sections (en gras italique) : la déclaration des dispositifs composée de descripteurs, les connections,

et la partie présentation, optionnelle et propre à l'éditeur de configurations. Chaque descripteur est nommé (en gras) dans la première section afin de pouvoir être référencé dans les sections suivantes. Une syntaxe similaire permet également le nommage des descripteurs de slots, lorsque des connexions doivent être spécifiées entre des slots dont le nom concret n'est pas connu (attribut `name` absent du descripteur ou comportant des caractères génériques, par exemple).

L'auteur d'une configuration peut éditer ce fichier et modifier les descripteurs qui s'y trouvent avant de le distribuer à d'autres utilisateurs. Par exemple, les slots non utilisés d'un dispositif peuvent être supprimés de sa déclaration, ou plusieurs dispositifs interchangeables peuvent être spécifiés en reliant simplement leurs descripteurs par un `or`. Les opérations ensemblistes `and`, `or` et `not` peuvent être imbriquées et appliquées sur n'importe quel sous-ensemble de descripteurs.

Le langage ICSCRIPT est un premier pas vers la réutilisabilité des configurations d'entrée. À terme, des stratégies de réutilisabilité intégrées pourront être mises en œuvre sur la base des descripteurs, telles que la visualisation et l'édition graphique des descripteurs, ou l'utilisation d'un moteur de résolution de contraintes dans l'algorithme de chargement des configurations.

4.4 Construction et édition de configurations d'entrée

L'éditeur de configurations d'ICON permet de visualiser, modifier, sauver et charger des configurations d'entrée. La construction et l'édition des configurations d'entrée se font essentiellement par manipulation directe. Bien que les configurations d'entrée soient indépendantes de l'éditeur et peuvent être implémentées directement en Java, cet outil constitue l'élément central de la boîte à outils ICON, et lui confère toute sa souplesse. Le développeur d'applications peut, à travers cet outil, construire et tester un grand nombre de configurations d'entrée potentielles qui répondent à des situations spécifiques en termes de dispositifs d'entrée. L'utilisateur avancé peut ensuite personnaliser l'interaction pour la plier à ses besoins spécifiques.

Dans cette section, nous présenterons brièvement l'éditeur de configurations d'ICON. Puis, nous donnerons aperçu des différents types de configuration d'entrée qu'il est possible de construire avec cet éditeur. Nous avons construit et testé de nombreuses méthodes d'entrée avec ICON, et nous n'en fournirons que les exemples les plus représentatifs.

4.4.1 L'éditeur de configurations

L'efficacité d'un outil de programmation visuelle repose en grande partie sur le choix des techniques d'interaction, choix qui détermine sa facilité d'apprentissage ainsi que les performances effectives dans les différentes tâches de programmation. Les techniques employées dans notre éditeur, simples et standards pour la plupart et plus spécifiques pour d'autres, répondent aux tâches courantes de construction et d'édition de configurations d'entrée. Nous les évoquons dans ce court descriptif. Des clips vidéo montrant des constructions ou des modifications de configurations sont également disponibles en téléchargement [DRAGICEVIC 02].

La fenêtre d'édition. Une capture d'écran de la fenêtre d'édition de configurations d'entrée est donnée figure 4.12 page 129. Celle-ci est divisée en trois parties : En haut à gauche, l'arborescence des dossiers de dispositifs-prototypes, qui comprend les dispositifs d'entrée disponibles, les dispositifs utilitaires d'ICON, et le cas échéant, le dossier de l'application en cours d'édition. En bas à gauche,

```
Devices {
  mouse: and {
    folder=all.extended.directInput;
    and {
      name=mouse;
      class=DDirectMouse;
      externalInput;
      outs {
        and {
          name=but.left;
          type=boolean;
        }
        and {
          name=but.middle;
          type=boolean;
        }
        and {
          name=but.right;
          type=boolean;
        }
        and {
          name=p.dx;
          type=int;
        }
        and {
          name=p.dy;
          type=int;
        }
        and {
          name=wheel;
          type=int;
        }
      }
    }
    properties disableSystem=yes;
  }
  sum: and {
    folder=all.standard.control;
    class=DSum;
    properties defaultValue=0.0;
  }
  sum2: and {
    folder=all.standard.control;
    class=DSum;
    properties defaultValue=0.0;
  }
  freehand: and {
    and {
      name=freehand;
class=fr.emn.examples.icondraw.DFreehand;
    }
    properties {
      screenCoordinates=yes;
      activated=yes;
    }
  }
}
```

```
Connections {
  mouse.but.left=freehand.use;
  mouse.p.dx=sum.in;
  mouse.p.dy=sum2.in;
  sum.out=freehand.p.x;
  sum2.out=freehand.p.y;
}


Layout {
  sum2: {
    x=81.50220396714089;
    caption=sum;
    expand=yes;
    scale=0.7142857142857143;
    y=107.5312562612703;
  }
  sum: {
    x=80.19785614105392;
    caption=sum;
    expand=yes;
    scale=0.7142857142857143;
    y=94.05299539170508;
  }
  freehand: {
    x=147.29663394109403;
    caption=freehand;
    expandSlots=[p];
    scale=1.0;
    y=79.66643291257597;
  }
  mouse: {
    x=1.8408134642356373;
    caption=mouse;
    expandSlots=[but, p];
    scale=1.0;
    y=79.9273024777934;
  }
}
```

FIG. 4.11 – Fichier ICSCRIPT généré à partir de la configuration de la figure 4.21.

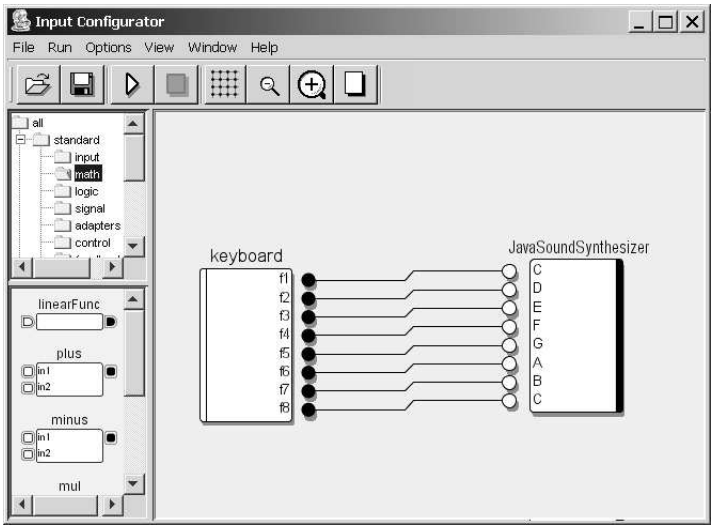


FIG. 4.12 – La fenêtre d'édition d'ICON. Une configuration qui permet de jouer de la musique avec les touche de fonction du clavier est en cours d'édition.

la liste des dispositifs-prototypes pour le dossier actif. À droite, l'espace de travail zoomable où est visualisée la configuration d'entrée.

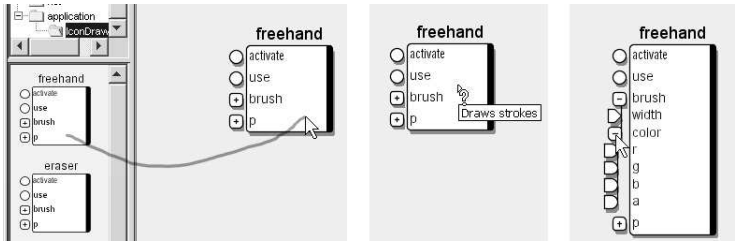


FIG. 4.13 – Instanciation (à gauche) et inspection (au centre et à droite) d'un dispositif.

Instanciation et inspection de dispositifs. Pour être utilisé dans une configuration, un dispositif est instancié à partir de son prototype par cliquer-glisser sur l'espace de travail (figure 4.13, image de gauche). Une technique de type fisheye peut être activée par un quasi-mode afin de « pousser » les dispositifs voisins si la place manque. Des info-bulles (*tooltips*) permettent d'obtenir des informations sur le dispositif : courte description du dispositif et de ses slots, nom complet et type de chaque slot, fenêtre d'aide détaillée sur le dispositif (figure 4.13 au centre). La hiérarchie des slots peut être inspectée à la manière des répertoires dans les explorateurs de fichiers (figure 4.13, image de droite).

Techniques de connexion. Les slots sont connectés par cliquer-glisser Les compatibilités de types et les cycles sont explicités par des retours graphiques et magnétiques. Le rendu des connexions est automatique et emploie un algorithme simple et original, optimisé pour les graphes acycliques[2]. Des techniques d'interaction dédiées permettent de faciliter certaines tâches de connexion courantes. L'*auto-expand* est une technique gestuelle qui ouvre et ferme automatiquement les slots composites

[2]Lors de déplacements de dispositifs, cet algorithme offre une animation fluide et sans discontinuité des connexions et ne génère pas de nouveau croisement.

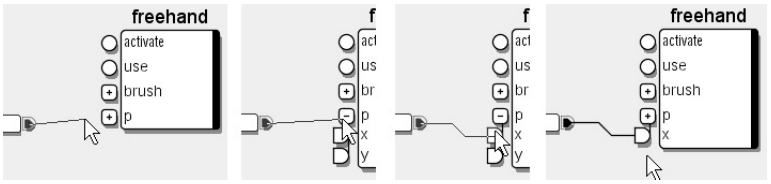


FIG. 4.14 – La technique *auto-expand*. Ici, un dispositif est connecté à un slot initialement invisible, en un seul geste.

pendant une interaction de connexion (figure 4.14)[3]. Une technique de *reconnexion* permet également de déplacer l'une des deux extrémités d'une connexion, et des *connexions groupées* peuvent être créées automatiquement entre des slots composites.

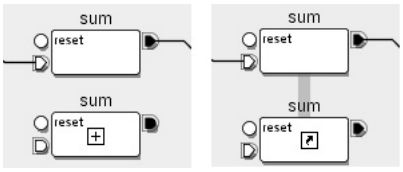


FIG. 4.15 – Création par manipulation directe d'une copie de dispositif (à gauche) et d'un lien (à droite).

Copie et lien. La copie d'un dispositif (figure 4.15, image de gauche) et la création d'un lien (figure 4.15, image de droite) se font par manipulation directe. L'opération de copie crée une nouvelle instance du dispositif et en reproduit le paramétrage. Les *liens*, explicités par des lignes verticales épaisses (figure 4.15, image de droite), permettent d'afficher le même dispositif en différents endroits de la configuration, afin de réduire sa complexité visuelle. Comme nous le verrons dans certaines configurations-exemples, les liens peuvent être déployés dans la dimension verticale, perpendiculairement au flux de connexions qui est essentiellement horizontal.

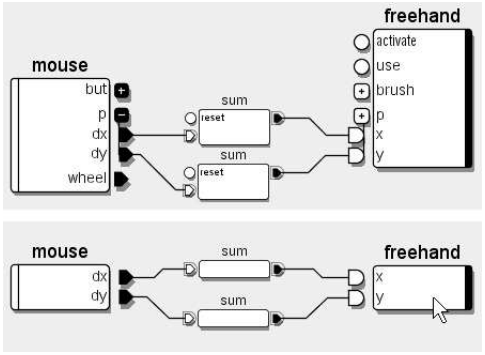


FIG. 4.16 – Configuration d'entrée avec dispositifs déployés (en haut) et réduits (en bas).

Niveau de détail. Les dispositifs peuvent être réduits pour ne montrer que les *slots utilisés*, et diminuer ainsi la complexité visuelle d'une configuration. Le passage d'un mode à l'autre se fait

[3]La fermeture automatique des slots composites est temporairement désactivée lors d'une inspection de haut en bas, afin d'éviter que le slot sous le pointeur se décale brutalement vers le haut.

par un simple clic sur le dispositif, ou automatiquement par la technique de l'*auto-expand* décrite précédemment. L'espace de travail est *zoomable*, ainsi que les dispositifs individuels (pas de zoom sémantique pour l'instant, en dehors d'une prise en charge minimale des niveaux de détail).

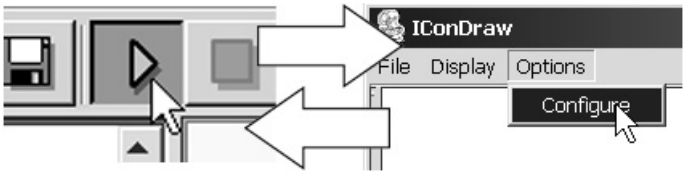


FIG. 4.17 – Cycle d'édition-exécution pour une configuration d'entrée.

Cycle d'édition-exécution. Une fois construite, une configuration d'entrée peut être immédiatement testée en lançant son exécution (figure 4.17, à gauche). Par la suite, la configuration peut à nouveau être éditée par l'emploi du raccourci `Alt+C` ou par une option de menu si l'application interactive le prévoit (figure 4.17, à droite). L'édition dynamique permet d'affiner une configuration d'entrée par des cycles d'édition-exécution, ce qui est particulièrement utile pour tester des techniques d'interaction, ou régler des paramètres arbitraires comme des niveaux de sensibilité.

4.4.2 Des configurations standard aux configurations hautement interactives

Dans cette partie, nous donnons un aperçu des techniques d'interaction qui peuvent être construites avec l'éditeur de configurations. Nous évoquerons des exemples de méthodes d'entrée adaptées à des configurations matérielles standard, enrichies ou appauvries, puis des configurations exploitant des techniques d'interaction novatrices et avancées. Tous les exemples sont issus de captures d'écran de l'éditeur d'ICON, ou de l'application contrôlée.

Description de méthodes d'entrée standards

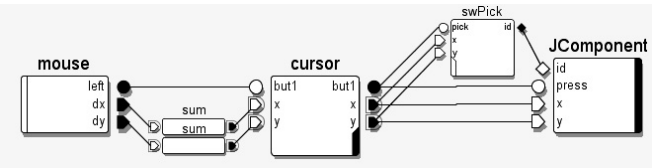


FIG. 4.18 – Contrôle positionnel standard des composants Swing.

Des méthodes d'interaction standards souris/clavier peuvent être construites et livrées avec les applications sous forme de configurations d'entrée par défaut. La figure 4.19 page suivante illustre une configuration qui reproduit le contrôle positionnel standard des composants de Swing à la souris. Les positions relatives de la souris bas-niveau sont transformées en valeurs absolues qui déplacent un curseur, puis sélectionnent le composant cible et contrôlent ce composant en générant des événements souris.

Cette configuration peut être étendue pour une description plus fine du comportement standard de Swing. La figure 4.18 met en évidence tous les slots disponibles sur les dispositifs de la figure 4.19. Par

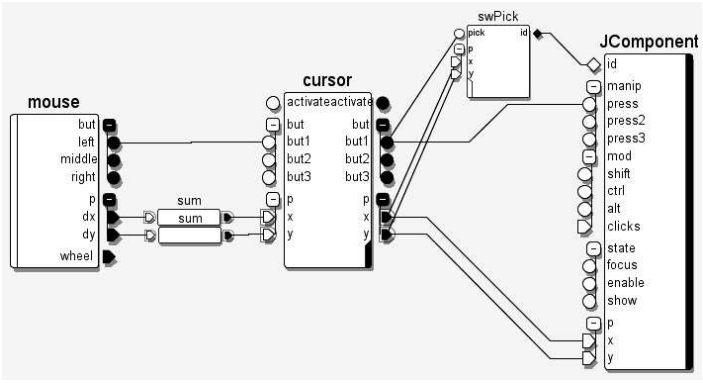


FIG. 4.19 – Contrôle positionnel standard des composants Swing. Sur cette figure, tous les slots disponibles sont mis en évidence.

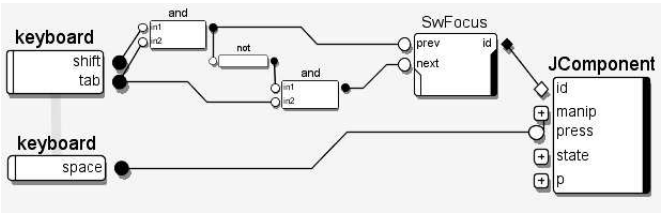


FIG. 4.20 – Contrôle au clavier du focus et des composants Swing.

des connexions supplémentaires, il est possible de prendre en compte les autres boutons de la souris et leurs modificateurs. Le slot *but1* du curseur peut être connecté au slot *state.focus* de *JComponent* pour décrire le changement de focus à la souris. La configuration peut également être complétée pour prendre en compte les mécanismes relatifs au clavier : la figure 4.20 montre un exemple de contrôle du focus avec les touches *Tab* et *Shift+Tab*, ainsi que la simulation du clic avec la touche *espace*.

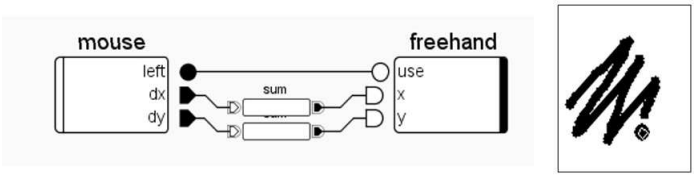


FIG. 4.21 – Contrôle à la souris du dessin à main levée.

Des méthodes d'interaction standard, basées sur la souris et le clavier peuvent également être construites pour des applications spécifiques. Sur la figure 4.21, l'outil de dessin à main levée est contrôlé à la souris de manière triviale. Pour être complète, cette configuration minimale peut être étendue en insérant un feedback supplémentaire sur le pointeur, et en décrivant le contrôle des attributs de brosse, ainsi que le contrôle modal d'outils multiples.

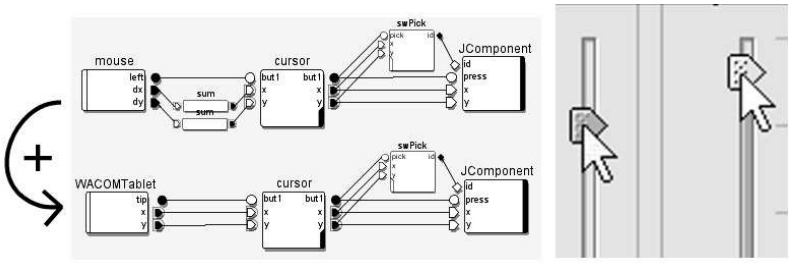


FIG. 4.22 – Interaction multi-pointeurs avec Swing.

Pointeurs multiples et interaction bimanuelle

Parce que chaque curseur gère son propre retour graphique, l'utilisation de curseurs multiples est élémentaire avec ICON. Comme illustré sur la figure 4.22, la duplication de la configuration de la figure 4.18 page 131 et le remplacement de la souris par un autre dispositif suffit à permettre un contrôle bimanuel ou multi-utilisateurs des applications Swing. Deux composants interactifs (ou plus, selon le nombre de pointeurs) peuvent être contrôlés en même temps. Cependant les interactions concurrentes ne sont pas permises au sein d'un même composant. En outre, certains mécanismes de Swing, conçus sur la base d'un pointeur unique, peuvent se révéler gênants : par exemple, le fait de cliquer avec un pointeur peut fermer un menu qui vient d'être ouvert par un autre pointeur. Nous discuterons de ce problème dans la section 5.6.2 page 170

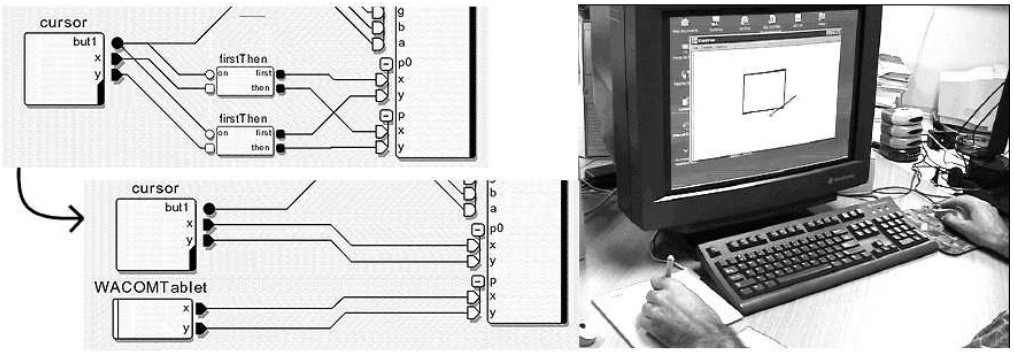


FIG. 4.23 – Tracé de lignes standard et bimanuel avec ICONDraw.

Contrairement aux dispositifs Swing, les dispositifs d'applications peuvent être conçus pour gérer la concurrence, et des techniques d'interaction bimanuelle peuvent être décrites avec une granularité bien plus fine. La figure 4.23 montre comment, avec un dispositif supplémentaire, une technique d'interaction standard peut être remplacée par une technique d'interaction bimanuelle. Le cadre supérieur gauche illustre le contrôle par défaut de l'outil de tracé de lignes d'ICONDraw, construit avec des dispositifs de routage conditionnel. Cette technique consiste à placer successivement une extrémité de la ligne puis l'autre. Dans le cadre inférieur, la configuration a été modifiée afin que le curseur (contrôlé par la souris) soit directement relié à une extrémité de la ligne, et qu'une tablette graphique soit reliée à la deuxième.

Utilisation de dispositifs étendus et riches

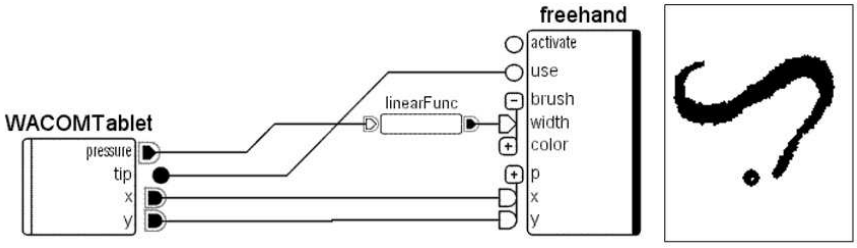


FIG. 4.24 – Dessin à main levée sensible à la pression.

Dans ICON, les dimensions supplémentaires fournies par les dispositifs étendus peuvent être facilement exploitées. La figure 4.24 montre comment, dans une configuration ICONDraw, le canal de pression d'une tablette graphique peut être assigné à la dimension de la brosse.

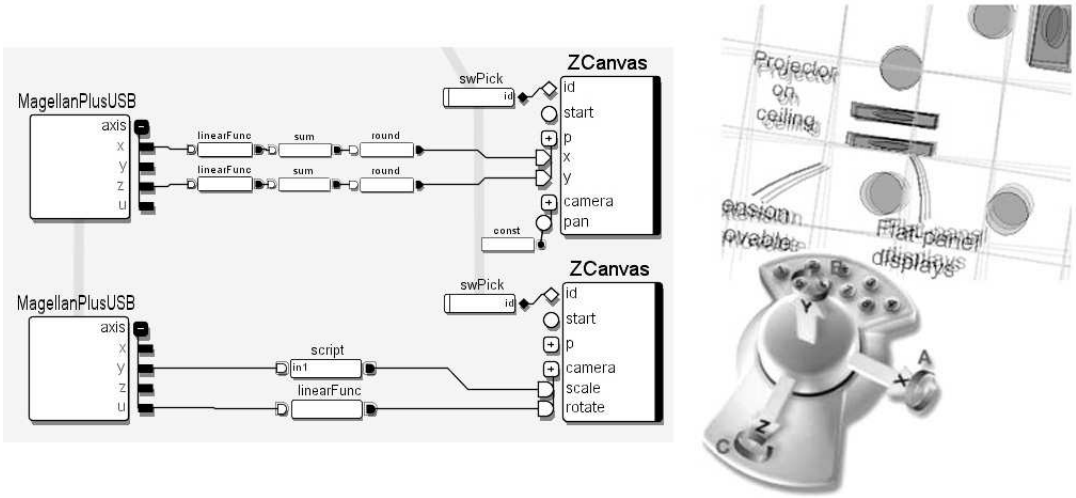


FIG. 4.25 – Contrôle d'une interface zoomable avec un dispositif à six degrés de liberté, par la technique *PZR*.

Les dispositifs d'entrée riches, comme les dispositifs à multiples degrés de liberté, peuvent également être exploités à leur maximum. La figure 4.25 illustre une configuration où une application zoomable, décrite par le dispositif *ZCanvas*, est contrôlée par un dispositif isométrique à six degrés de liberté de type *Magellan*. La technique employée (que nous nommons PZR pour « Pan/Zoom/Rotate ») permet de contrôler simultanément la position, le zoom et l'orientation du document.

Dans cette configuration-exemple, les dimensions *x* et *z* du dispositif, parallèles aux axes de la souris, déplacent la zone de travail. Sur chaque axe, trois dispositifs de traitement enchaînés convertissent des valeurs relatives réelles en valeurs absolues entières. La dimension *y* du dispositif, qui traduit la pression ou la traction verticale sur le dispositif, envoie des commandes de zoom au document à travers un dispositif scriptable, dont le code a été donné en exemple dans la section 4.3.1 page 118. Enfin, la dimension *u* envoie des commandes de rotation au document, à travers une simple remise à l'échelle.

4.4.3 Configurations d'entrée pour l'accessibilité

Émulation de dispositifs

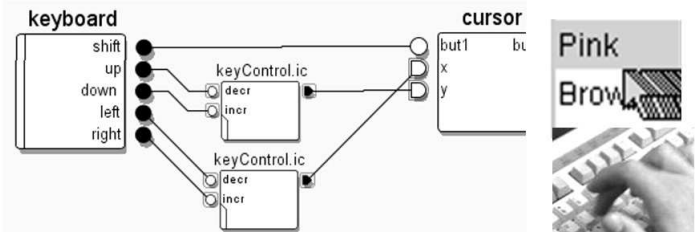


FIG. 4.26 – Adapteurs de contrôle clavier.

Deux dispositifs compatibles (par exemple, deux dispositifs positionnels) peuvent être facilement intervertis dans ICON. Si les dispositifs sont de nature différente, la puissance d'expression d'ICON peut être exploitée pour construire des techniques d'interaction capables d'émuler un dispositif. La bibliothèque d'ICON fournit en outre des adaptateurs prédéfinis pour ce type de tâche. La figure 4.26 montre comment un tel adaptateur peut être inséré entre un clavier et un curseur : l'adaptateur *Key-Control* est un dispositif composite construit avec ICON, qui permet un contrôle fluide de dimensions continues à partir de dispositifs à deux états. Des techniques plus spécifiques de contrôle adaptées à des entrées appauvries peuvent également être construites. Des touches clavier peuvent par exemple être directement assignées à des boutons ou à des commandes dans une application graphique.

Utilisation de commandes vocales

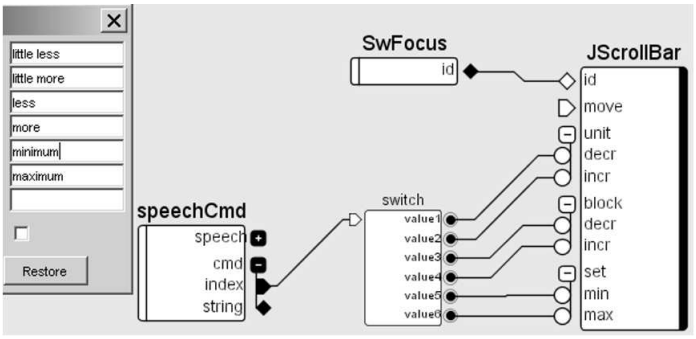


FIG. 4.27 – Contrôle vocal des barres de défilement dans Swing.

Bien qu'il ne gère pas les grammaires, le dispositif de commande vocale peut être employé pour un contrôle vocal simple d'applications interactives. La figure 4.27 illustre une configuration permettant de manipuler les barres de défilement de Swing à partir de commandes telles que *plus*, *un peu moins*, ou *minimum*. Sur cette figure sont représentés la fenêtre de propriétés dans laquelle le vocabulaire est spécifié, ainsi que le dispositif de contrôle conditionnel *Switch* permettant d'activer une commande en fonction de l'index de la chaîne reconnue.

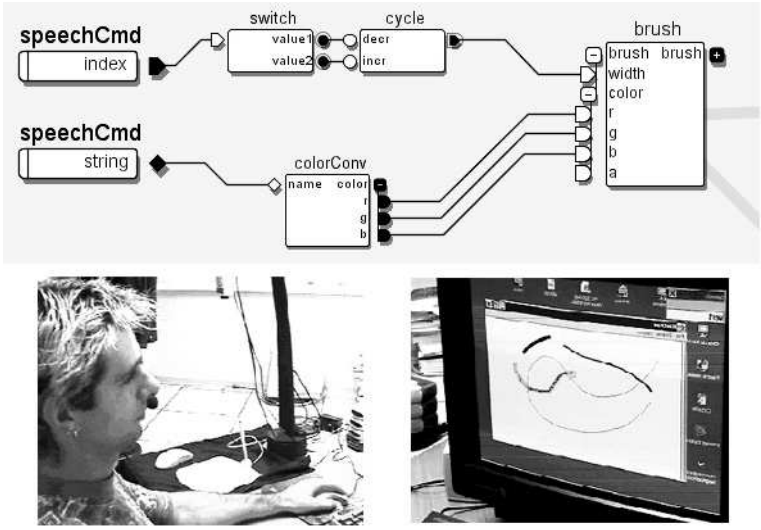


FIG. 4.28 – Les attributs de dessin contrôlés à la voix dans ICONDraw.

La figure 4.28 montre comment, dans ICONDraw, des attributs de brosse peuvent être contrôlés à la voix, avec des commandes comme *bleu*, *rouge*, *plus gros*, ou *moins gros*. La taille de la brosse est contrôlée par un dispositif de commande vocale suivi d'un dispositif *Switch*, puis d'un dispositif *Cycle* qui permet un contrôle incrémental paramétrable d'une valeur entière. Le contrôle de la couleur emploie un second dispositif de commande vocale paramétré avec un vocabulaire de couleurs. La chaîne de caractères reconnue est convertie en composantes RGB par un dispositif qui a été implémenté pour l'occasion (voir section 4.3.1 page 118).

Parmi d'autres mécanismes de contrôle vocal que nous avons expérimentés, une technique de curseur contrôlé à la voix, décrite dans la section 4.4.4, a été implémentée pour permettre un contrôle vocal générique des applications interactives.

4.4.4 Les techniques d'interaction avancées et novatrices

Concevoir de nouvelles techniques d'interaction : le pointage augmenté

Les dispositifs de traitement de données et de retour graphique d'ICON peuvent servir à construire des techniques d'interaction à part entière. Ces techniques d'interaction peuvent être spécifiques à une application, ou réutilisables si elles sont décrites à un niveau assez bas dans la configuration. Ces dernières peuvent être transformées en dispositifs composites et être réutilisées dans d'autres configurations.

Nous présentons en guise d'exemple un nouveau paradigme d'interaction conçu et testé avec ICON, le *pointage augmenté*. Le pointage augmenté décrit un ensemble de techniques d'interaction positionnelles qui emploient un double pointeur, composé d'un pointeur témoin (ou de bas niveau) et d'un pointeur augmenté (ou de haut niveau) (figure 4.29 page suivante). Le premier pointeur produit un retour graphique direct sur le dispositif positionnel sans agir sur l'interface. Le second pointeur, qui agit sur l'interface, est lié au premier mais possède un comportement supplémentaire destiné à

FIG. 4.29 – Le double pointeur, composé du pointeur témoin (en noir et blanc) et du pointeur augmenté (en gris foncé, à l'intérieur du précédent).

faciliter certaines tâches de pointage.

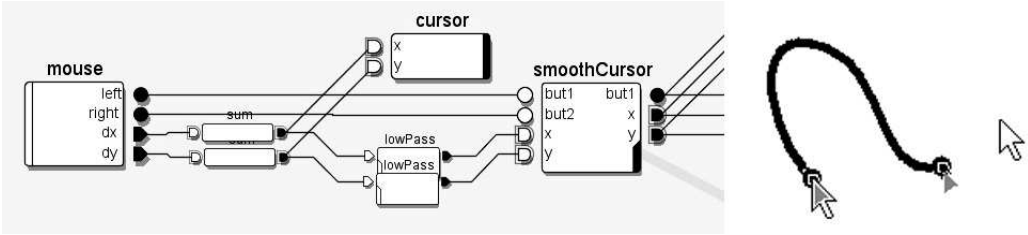


FIG. 4.30 – Une technique de pointage « lissé », permettant notamment la correction du dessin à main levée.

Pour des utilisateurs présentant des difficultés motrices ou dans certaines tâches positionnelles telles que le dessin à main levée, il peut être utile de lisser les déplacements du pointeur. La figure 4.30 présente la partie d'une configuration d'entrée décrivant cette technique appliquée au pointage augmenté : le premier dispositif *Cursor* constitue le pointeur témoin. Le second, renommé *smoothCursor*, placé à la suite de filtres passe-bas, constitue le pointeur augmenté. Ce dernier fait office de dispositif virtuel et contrôle d'autres parties de la configuration. Cette configuration a rapidement été obtenue en modifiant une configuration de contrôle positionnel standard. Sur la partie droite de la figure, la technique est utilisée pour le dessin à main levée dans ICONDraw : au départ et à la fin du tracé, les deux curseurs sont confondus. Pendant le tracé, le curseur effectif (en gris foncé) subit un retard et se désolidarise du premier pour tracer une ligne fluide.

Ici comme dans les exemples qui vont suivre, le pointeur témoin est utile pour la compréhension et la visualisation du comportement du pointeur augmenté par rapport aux mouvements réels du dispositif, tout en conservant une sensation de contrôle sur ce dernier.

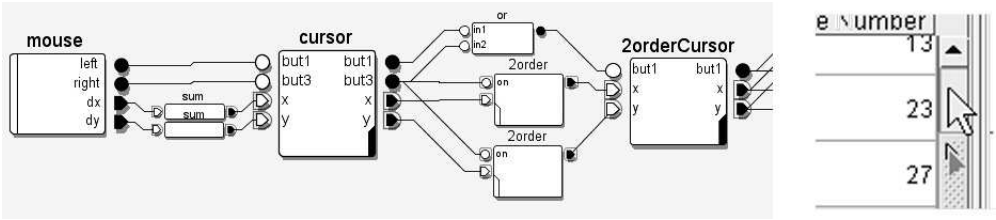


FIG. 4.31 – Le pointage de second ordre, appliqué à une barre de défilement : l'utilisateur agit uniquement sur la vitesse de défilement du document.

La figure 4.31 décrit une autre technique de pointage augmenté, le pointage de second ordre. Les filtres passe-bas sont remplacés par des dispositifs dérivateurs, qui calculent la vitesse d'évolution de valeurs numériques (ces derniers ont été décrits avec ICON puis transformés en dispositifs compo-

sites). Ici, le pointage augmenté est activé par le bouton droit de la souris, le bouton gauche effectuant un cliquer-glisser standard.

Le pointage de second ordre permet d'agir sur la vitesse du pointeur au lieu de sa position. Il est utile lors de tâches de pointage qui requièrent un déplacement uniforme et prolongé de celui-ci : recherche dans un long document, ou recherche dans une liste avec sélection successive des éléments. Il permet également des cliquer-glisser non limités aux bords de l'écran, utiles pour la navigation *(pan)* dans un large espace de travail. Sur la partie droite de la figure 4.31 page précédente, l'utilisateur met à contribution la technique sur une barre de défilement Swing[4]

D'autres techniques de pointage augmenté ont été mises en œuvre, comme le pointage contraint (une aide au tracé de lignes horizontales ou verticales nécessite uniquement un blocage conditionnel des valeurs en x ou y dans la configuration), ou la grille magnétique de bas-niveau. Toutes ces techniques géométriques permettent d'ajouter des fonctionnalités à des applications interactives sans modifier le code de celles-ci.

Pour finir, nous décrirons une technique assimilable à un pointage augmenté, le *curseur vocal*. Un pointeur est contrôlé à l'aide de commandes vocales agissant sur sa direction et sa vitesse, et deux autres commandes permettent de « simuler » l'appui et le relâchement du bouton de la souris. Le temps de reconnaissance, de l'ordre de la seconde pour la plupart des moteurs, pose d'importants problèmes pour le contrôle positionnel. Notre technique du curseur vocal permet de prendre en compte le *contexte* d'une commande, c'est-à-dire le moment où cette commande a commencé à être prononcée.

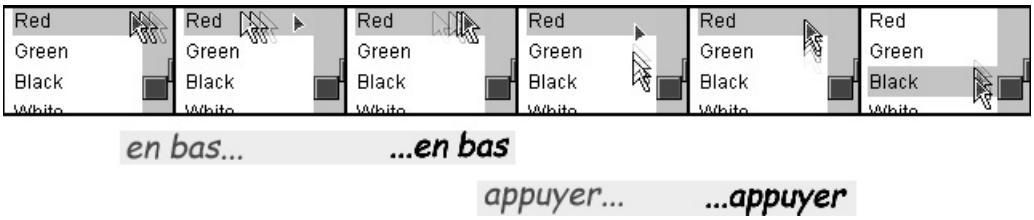


FIG. 4.32 – Exemple d'utilisation du curseur vocal. Les mots gris sont les mots prononcés, et les mots en noir sont les mots reconnus. Les images ont été retouchées pour rendre compte des mouvements du pointeur.

Le curseur vocal est composé d'un curseur effectif et d'un curseur contextuel (figure 4.32). En temps normal, ils se déplacent de façon solidaire. Lorsque l'utilisateur commence à parler (augmentation brutale du niveau d'entrée), le curseur contextuel s'arrête et le curseur effectif continue à se déplacer. Si un bruit ou une commande non-contextuelle (contrôle de la vitesse) est finalement reconnue, le curseur contextuel rejoint le curseur effectif. Si une commande contextuelle (changement de direction ou appui/relâchement) est reconnue, le curseur effectif rejoint le curseur contextuel et la commande est validée.

La configuration d'entrée de la figure 4.33 page ci-contre décrit le gros du mécanisme du curseur vocal. Un dispositif *SpeechCursor* effectue les principaux calculs en déterminant les positions successives des deux curseurs en fonction des commandes reçues. Le dispositif de commande vocale est configuré conformément aux commandes supportées par *SpeechCursor*, et est relié au dispositif de synthèse vocale afin d'avoir un retour sur les commandes reconnues. Dans cette variante de

[4]Nous avons également étendu le dispositif DJScrollbar pour qu'il puisse être manipulé avec une précision supérieure à celle du pixel. Le défilement d'un document peut ainsi être contrôlé finement par un pointage de second ordre mais aussi par tout dispositif de type tablette. L'affichage du curseur a également été modifié pour utiliser une interpolation linéaire.

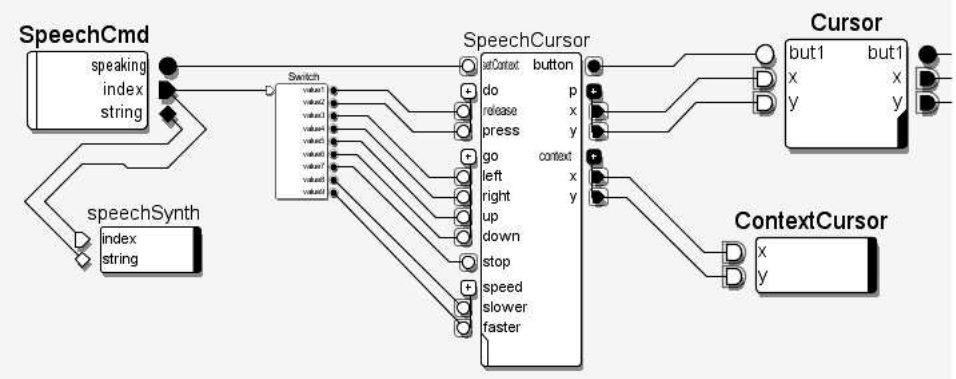


FIG. 4.33 – Une configuration décrivant la technique de pointage vocal.

pointage augmenté, les deux curseurs sont situés au même niveau dans la configuration. En effet, le déplacement de chacun des deux curseurs est fonction de l'autre.

Le dispositif *SpeechCursor*, principal dispositif de cette configuration, est un dispositif composite construit avec des dispositifs utilitaires de base. Celui-ci est relativement complexe : il comporte 17 dispositifs et emploie un dispositif composite lui-même constitué de 21 dispositifs de base. Nous y reviendrons dans la section 5.4 page 157 lorsque nous développerons le problème de la complexité.

Les techniques d'interaction avancées « clés en main »

Avec la technique du curseur vocal vue précédemment, nous ne sommes plus très loin des limites pratiques de la construction de comportements complexes à partir de briques de base. Pour décrire des techniques d'interaction complexes, l'emploi de dispositifs scriptables ou l'implémentation de dispositifs monolithiques sont des solutions à privilégier.

Les techniques d'interaction implémentées sous forme de dispositif, « prêtes à l'emploi », rendent possible la description rapide d'interfaces évoluées. La bibliothèque d'ICON comporte actuellement trois de ces techniques d'interaction, toutes trois basées sur la semi-transparence : la *Toolglass*, le *Floating Quikwriting*, et le dispositif de *Commande Gestuelle*.

Le dispositif *Toolglass* permet d'animer une barre d'outils semi-transparente, manipulable en interaction bimanuelle. Ce dispositif implémente la technique d'interaction de même nom (voir section 1.3.3 page 19), avec de plus une transparence contrôlable. La présentation visuelle de cette barre d'outils (liste et disposition des icônes à afficher) est spécifiée dans la fenêtre de propriétés du dispositif (figure 4.34 page suivante).

Les slots *x* et *y* de la toolglass ont fonction de *picking* : lorsque son slot *use* est à vrai, le dispositif émet vrai sur l'un des slots de *tool* selon l'outil qui se trouve à la position donnée. Cette position est également retransmise en sortie sur les slots *xthru* et *ythru*. Dans la configuration de la figure 4.34 page suivante, chaque slot booléen de *tool* est connecté au slot *activate* d'un curseur dédié (seul le contrôle de l'outil *Freehand* est représenté ici, les autres étant similaires). Lorsqu'aucune icône n'a été sélectionnée, le curseur actif est le curseur par défaut (en haut sur la figure). Dans le cas contraire, un autre curseur est activé (en bas sur la figure), et à travers lui, l'outil correspondant. Pour finir, la toolglass peut être déplacée à travers ses slots *xmove* et *ymove*, et sa transparence peut également être

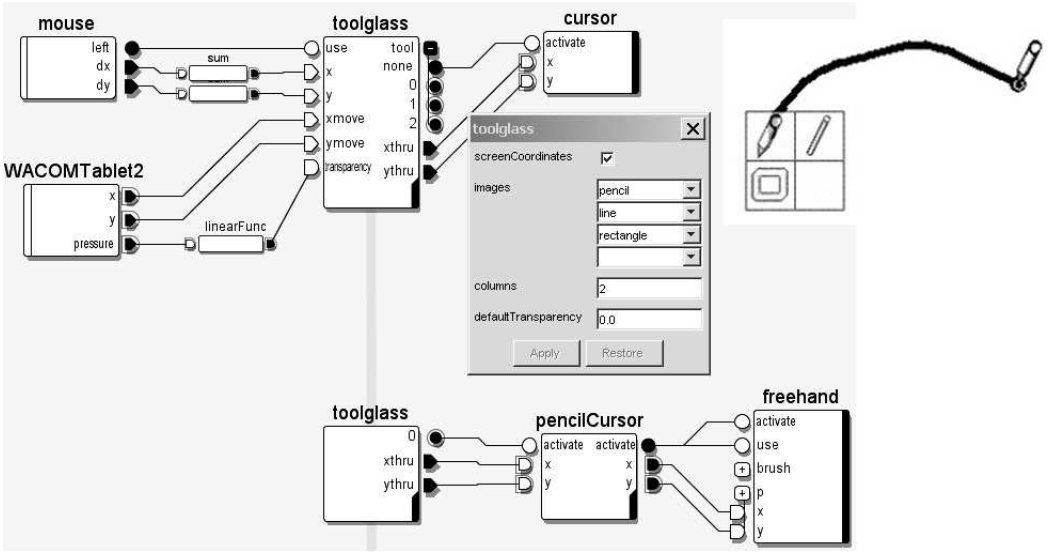


FIG. 4.34 – Branchement de la technique d'interaction *Toolglass* dans ICONDraw.

contrôlée. Ici, l'emplacement et la transparence de la barre d'outils sont simultanément contrôlées au stylet et les outils sont contrôlés à la souris.

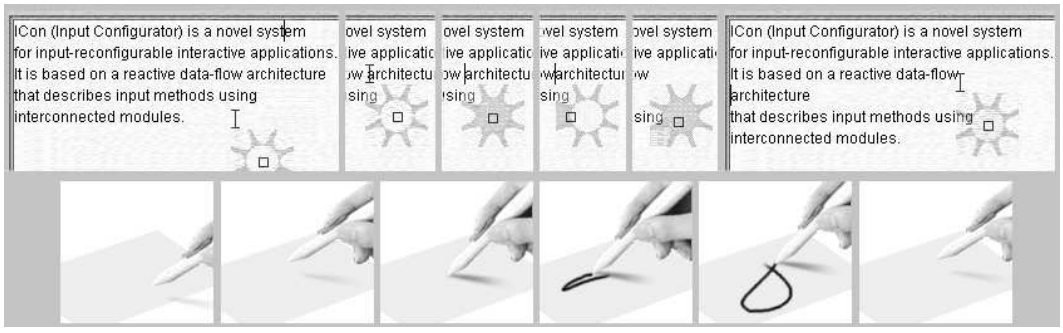


FIG. 4.35 – Scénario d'utilisation de la technique d'interaction *Floating Quikwriting* dans Swing. Ici, l'utilisateur supprime une espace et insère un saut de ligne en un geste rapide.

Le dispositif *QuikWrite* implémente une version semi-transparente de la technique d'interaction gestuelle de même nom (voir section 1.3.2 page 16), appelée *Floating QuikWriting*. Au lieu d'être fixe, la zone de saisie gestuelle est flottante et est solidaire d'un pointeur en forme de caret (figure 4.35). Dans le mode pointage, la position de l'ensemble est contrôlée par un dispositif positionnel. Lors du passage au mode gestuel (bouton enfoncé), le curseur textuel est déplacé à la position du pointeur, et l'objet flottant s'immobilise pour passer la main à son curseur gestuel interne.

Cette technique est efficace pour la révision de documents, car des éditions simples comme la suppression ou l'insertion d'un caractère dans le texte se font en un geste rapide, et parce qu'elle ne requiert pas de changement de dispositif entre la navigation et la saisie.

Le dispositif *QuikWrite* reçoit simplement des événements positionnels et les transcrit en com-

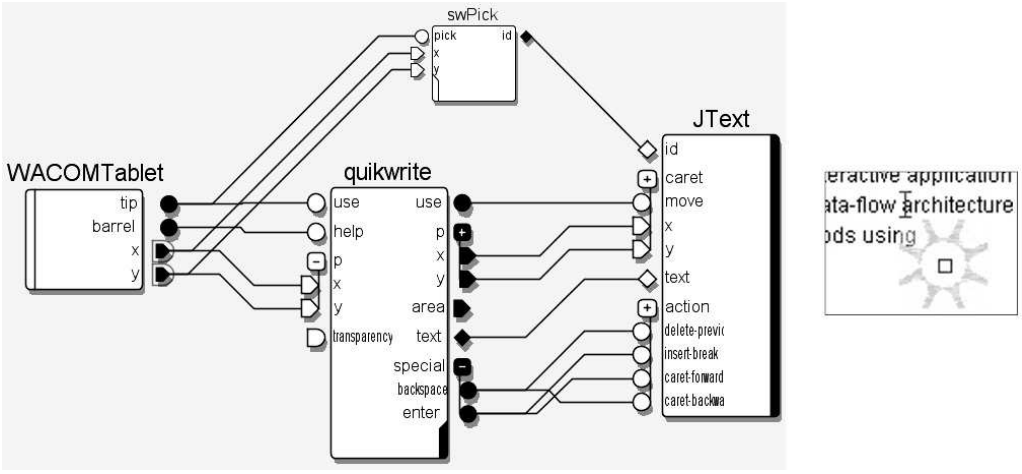


FIG. 4.36 – Configuration d'entrée pour l'utilisation du *Floating Quikwriting* dans Swing.

mandes booléennes, en plus de produire un retour graphique. La configuration de la figure 4.36 illustre son utilisation avec les composants texte de Swing.

Implémenté récemment, le dispositif *GestureCmd* offre une prise en charge de l'interaction gestuelle conventionnelle. Il repose sur le moteur de classification de Satin [HONG & LANDAY 00] dont il emploie également les vocabulaires prédéfinis et l'interface d'entraînement. La fenêtre de propriétés du dispositif permet de lancer celle-ci, et de paramétrer l'aspect de la trace affichée en incrustation. Le dispositif de commandes gestuelles émet, en plus des commandes et des classes de commandes reconnues, des propriétés géométriques telles que le rectangle englobant, le point de départ et le point d'arrivée du geste. Nous avons employé ce dispositif pour dessiner des formes géométriques dans ICONDraw et pour saisir du texte dans les applications Swing avec l'alphabet Graffiti [MACKENZIE & ZHANG 97].

4.5 Distribution, contributeurs, et projets utilisant ICON

La version actuelle de la boîte à outils ICON comporte approximativement 400 classes (dont la moitié est employée pour décrire les dispositifs de la bibliothèque et leurs processeurs spécialisés) et approximativement 20 000 lignes de code (dont un tiers est consacré à l'éditeur interactif). Elle a été développée par l'auteur, exceptées certaines parties, en particulier l'application ICONDraw développée par Jean-Daniel Fekete, les dispositifs *GestureCmd* et *VirtualUser* implémentés par Stéphane Huot, et les dispositifs de la *X Input Extension* implémentés par Stéphane Conversy. ICON repose en outre sur un bon nombre d'APIs d'entrée pour la prise en charge des dispositifs physiques (voir la section 4.2.1 page 111 pour plus de détails).

Une distribution en version alpha d'ICON est en libre téléchargement à l'adresse suivante : `http://www.emn.fr/x-inf`

En dehors des applications-jouet évoquées dans ce chapitre, ICON est actuellement utilisé pour la conception de deux applications interactives dans le cadre du projet *GINA* (Géométrie Interactive et Naturelle) [CMI 02]. Ce projet a pour objectif de fournir des outils de reconstruction 3D à partir de vues 2D en perspective et de contraintes spécifiées de manière multimodale. Les applications sont les

suivantes :

- **Marina II** [CMI 02] est une application de reconstruction de scènes 3D à partir de photographies basé sur le solveur de contraintes GINA. Développée avec Swing, cette application comporte déjà un premier niveau de configurabilité à travers les dispositifs Swing d'ICON. Elle externalise en outre des dispositifs simples de type « commande » afin de permettre la configuration des actions. Par exemple, les changements de vue peuvent être contrôlés avec le bouton droit de la souris ou par des commandes vocales.
- **Svalabard** [CMI 02] est un projet d'interface Post-WIMP de modélisation 3D par croquis, employant une approche centrée utilisateur et se basant notamment sur des études portant sur les techniques de dessin d'architectes [HUOT *et al.* 03]. Cette application emploie actuellement une tablette graphique et des filtres pour construire et traiter les traces, et détecter le contexte du dessin (figure 4.37). Ces filtres sont des dispositifs ICON, ce qui permet de décrire et tester aisément des mécanismes de filtrage et les affiner dynamiquement.

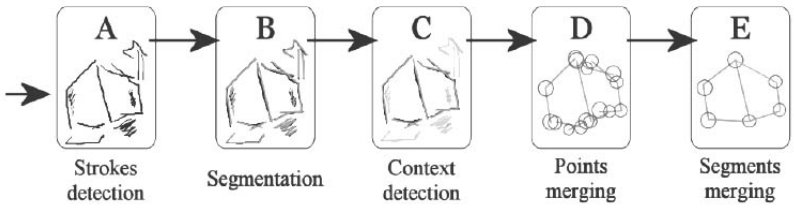


FIG. 4.37 – Filtres de dessin dans *Svalabard* [HUOT *et al.* 03].

ICON est également employé dans un projet du LRI visant à évaluer l'efficacité des techniques d'interaction Post-WIMP pour différents types de tâches [APPERT *et al.* 03]. Il a servi dans ce projet à monter une expérimentation contrôlée où plusieurs techniques de sélection d'outils (toolglass, palette d'outils classique et palette d'outils bimanuelle) sont comparées dans des tâches simples.

Il existe pour finir deux projets de boîtes à outils graphiques reposant sur ICON :

- **PiccoloIcon :** Nous avons déjà expérimenté quelques techniques d'interaction avec la boîte à outils zoomable *Jazz* [BEDERSON *et al.* 00], dont une que nous avons évoqué dans ce chapitre. Des travaux sont actuellement en cours pour intégrer ICON de manière complète et raisonnée à la boîte à outils *Piccolo* [BEDERSON 03], le successeur de *Jazz*.
- **MagLite** [HUOT 03] est une boîte à outils graphique Post-WIMP en cours de développement par l'auteur de *Svalabard,* reposant intégralement sur ICON pour la gestion des entrées. Elle permet de construire aisément des objets interactifs de forme quelconque et manipulables en translation, taille et rotation avec ICON. Elle permet également de décrire et d'associer des outils à ces composants. Bien que cette boîte à outils ne soit qu'en cours de réalisation, la palette des techniques d'interaction Post-WIMP connues ou inédites pouvant être décrites est déjà très vaste.

4.6 Conclusion

Nous avons décrit ICON, une boîte à outils d'entrée dont l'objectif est de résoudre les principaux problèmes liés à l'adaptabilité en entrée actuellement non abordés par les boîtes à outils graphiques.

Nous avons principalement montré à travers des exemples qu'ICON permettait de construire aisément une grande variété de techniques d'interaction Post-WIMP et d'accessibilité, existantes ou inédites, et que ces techniques une fois construites étaient extrêmement configurables.

La boîte à outils en entrée ICON est un projet vaste, qui a mis longtemps à atteindre la maturité nécessaire pour commencer à être utilisé dans des applications grandeur nature. Les utilisateurs ayant employé ou employant ICON sont pour l'heure très satisfaits de cet outil, car il leur ouvre des possibilités que nul autre n'est actuellement en mesure de fournir. Ces utilisateurs ont également, par leurs remarques, grandement contribué à son amélioration. En distribuant librement ICON, nous espérons à la fois populariser notre approche dans le monde du développement et de la recherche, la valider par un nombre plus important d'applications, et obtenir des retours constructifs afin d'améliorer ICON.

ICON fournit des solutions inédites à la plupart des problèmes liés à l'interaction en entrée, mais ne prétend pas les résoudre tous. Il ne prétend pas non plus à l'universalité. Dans le chapitre suivant, nous analyserons les forces et les faiblesses de notre approche en général et d'ICON en particulier, et tenterons également de répondre aux nombreuses questions qu'elles ouvrent.

Chapitre 5

Discussion

Sommaire

5.1 Introduction

Dans ce chapitre, nous identifions les apports essentiels d'ICON, ainsi que ses limites, à la fois celles qui sont inhérentes à notre modèle et celles pour lesquelles nous pensons pouvoir proposer des solutions. Nous généraliserons également quelques aspects de notre approche.

Dans un premier temps, nous expliquons comment ICON étend l'architecture des systèmes interactifs actuels pour rendre la gestion des entrées explicite. Puis, nous tentons de déterminer dans quelle mesure ICON et son modèle sous-jacent sont pertinents pour décrire les paradigmes d'interaction non conventionnels évoqués dans le premier chapitre, et poursuivons avec les problèmes de complexité et de lisibilité relatives au langage visuel.

Nous généralisons et détaillons également le rôle des différents types de dispositifs, et en particulier les services que doivent fournir à ICON les dispositifs de boîte à outils et d'application. Nous analysons ensuite la pertinence d'ICON du point de vue du programmeur, en abordant différentes considérations d'ordre pratique. Puis, nous évoquons les différents utilisateurs auxquels ICON s'adresse.

Pour finir, nous comparons ICON avec les approches existantes et proposons quelques perspectives, avant d'aborder la conclusion générale de cette thèse.

5.2 Les apports d'ICON du point de vue architectural

ICON repose sur des systèmes interactifs conventionnels (système d'exploitation + boîte à outils), systèmes dont nous avons déjà décrit les limites dans le chapitre 1 (section 1.4.3 page 30). Mais à l'inverse de ces systèmes il permet de décrire la manière dont les entrées sont gérées dans un modèle clair et consistant. Dans cette section, nous identifions les différents niveaux de gestion des entrées dans les systèmes interactifs réels, puis nous montrons la façon dont ICON parvient à étendre ce type de système, et ce que cette extension apporte.

5.2.1 Architecture concrète des systèmes interactifs

Dans les systèmes interactifs actuels, les applications s'appuient essentiellement sur des services partagés fournissant une gestion standard des entrées. Dans ces systèmes, les techniques d'interaction standard sont implémentées sur plusieurs niveaux (figure 5.1 page suivante) :

- **Niveau matériel :** Les entrées peuvent être gérées en partie au niveau matériel, par l'électronique du dispositif. C'est le cas par exemple des boutons de tir automatique sur certaines manettes de jeux, similaires à la fonction de répétition de touches du clavier. Ces mécanismes peuvent être vus comme des techniques d'interaction.
- **Niveau système :** Au niveau du système d'exploitation, les *pilotes* de dispositifs fournissent au programmeur des abstractions de bas niveau pour lire les données en provenance des dispositifs. Les pilotes ont également pour charge de transformer les données spécifiques en informations interprétables par le gestionnaire d'entrées standard du système d'exploitation. Certains pilotes de dispositifs non standard permettent ainsi un contrôle par compatibilité en imitant le comportement d'un clavier ou d'une souris. Le *gestionnaire d'entrées* standard du système d'exploitation fournit des services relatifs aux dispositifs standards, tels que l'affichage d'un pointeur

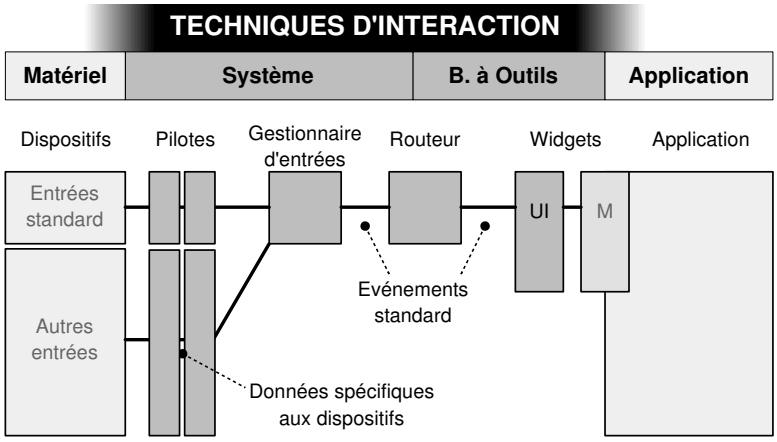


FIG. 5.1 – Architecture concrète d'un système classique, mettant en évidence les différents niveaux d'implémentation des techniques d'interaction standard.

ou la gestion de la langue du clavier, et produit les événements propres à ces dispositifs. Un premier *routage* de ces événements est effectué par le gestionnaire de fenêtres, qui les transmet à la fenêtre concernée.

– **Niveau boîte à outils :** La boîte à outils graphique sur laquelle repose l'application interactive se charge, pour chaque fenêtre, de router les événements vers les widgets appropriés. Il existe deux stratégies standard de routage : le multiplexage spatial pour les événements positionnels, et le multiplexage temporel basé sur les techniques de focus. Ces stratégies de routage constituent des techniques d'interaction. Chaque widget de la boîte à outils gère à son tour une technique d'interaction générique. Dans les boîtes à outils qui suivent une approche MVC, le *modèle* d'un widget est distinct de sa vue et de son contrôleur (regroupés dans un objet *UI* dans le langage Java). Dans cette approche, les objets *modèle* peuvent être considérés comme des briques de base de l'application.
– **Niveau applicatif :** Si les modèles conceptuels d'interaction recommandent une séparation franche entre l'interaction et le noyau applicatif, cette séparation est difficile à mettre en œuvre en pratique. En outre, les limitations inhérentes aux boîtes à outils obligent souvent l'implémentation de techniques d'interaction spécifiques au niveau de l'application.

5.2.2 Les niveaux d'accès et de contrôle d'ICON

La figure 5.2 page ci-contre montre comment ICON s'insère dans un système interactif concret. ICON est une boîte à outils d'entrées qui s'interpose entre le système d'exploitation et l'application. Lorsqu'il est intégré à une boîte à outils graphique comme Swing, il s'insère entre ses mécanismes de routage événementiel et ses widgets. ICON peut servir à étendre la gestion standard des entrées, mais est également capable de gérer la quasi-totalité de l'interaction des dispositifs d'entrée à l'application. Le rôle joué par ICON dans la gestion des entrées dépend essentiellement de la manière dont il accède aux entrées et de la façon dont il contrôle l'application.

ICON *accède aux entrées* à deux niveaux :

– **Accès aux événements standard :** ICON peut être contrôlé par des événements standard, à

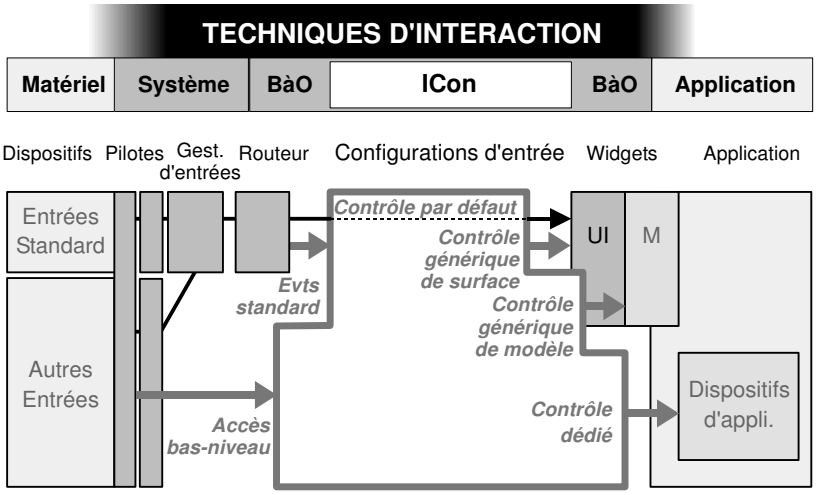


FIG. 5.2 – Un système interactif concret intégrant ICON.

travers des dispositifs qui lisent dans la file d'événements du système d'exploitation ou de la boîte à outils. Ces derniers possèdent l'avantage d'être portables d'une plate-forme à une autre. Le dispositif clavier de Swing (voir section 4.2.1 page 111) en est un exemple.

– **Accès bas niveau aux dispositifs :** ICON accède principalement aux dispositifs d'entrée au *niveau bas*, ce qui permet de tirer parti des spécificités des dispositifs non conventionnels, mais également de celles des dispositifs standard. Ainsi, un dispositif de souris bas-niveau peut émettre des données positionnelles non contraintes aux bords de l'écran (voir section 4.2.1 page 111).

ICON peut contrôler une application interactive de quatre manières différentes. Les *niveaux de contrôle* d'une application, du plus superficiel au plus profond, sont les suivants :

– **Le contrôle par défaut :** Ce niveau désigne le contrôle d'une application « hors ICON ». ICON permet de conserver la gestion des entrées effectuée implicitement au niveau système et boîte à outils, et de l'étendre ou non avec des interactions décrites explicitement. Par exemple, une application de modélisation 3D conventionnelle peut être étendue pour prendre en compte la manipulation 3D avec des dispositifs dédiés. Les différents mécanismes standard de gestion des entrées peuvent également être désactivés (voir section 4.2.3 page 114) pour être redéfinis de façon explicite ou être remplacés par des techniques d'interaction alternatives.
– **Le contrôle générique de surface :** Ce niveau désigne les stratégies de contrôle positionnel générique des widgets de la boîte à outils, à travers des dispositifs *manipulateurs de surface* (section 4.2.3 page 114). C'est à ce niveau que peuvent être explicitement redéfinies des techniques d'interaction standard telles que le *picking*. Certaines configurations décrites dans notre chapitre sur la boîte à outils ICON constituent des exemples de contrôle générique de surface : le contrôle positionnel standard des composants Swing (figure 4.18 page 131), le contrôle au clavier du focus et du clic (figure 4.20 page 132), et l'interaction Swing avec des pointeurs multiples (figure 4.22 page 133).
– **Le contrôle générique de modèle :** Ce niveau comprend les stratégies de contrôle spécifiques à un type de widget, décrites à travers des dispositifs *manipulateurs de modèle* (section 4.2.3 page 114). Avec le niveau précédent, il permet de spécifier des stratégies d'accessibilité géné-

riques compatibles avec les applications existantes. Le contrôle de modèle est générique en ce sens qu'il n'est pas dédié à une application particulière, mais il est cependant plus spécifique que le précédent, car il décrit des interactions avec une classe de widget donnée. La configuration de la figure 4.27 page 135, qui décrit le contrôle des barres de défilement à la voix, en est un exemple. Le contrôle d'une interface zoomable (figure 4.25 page 134) est un exemple d'utilisation de manipulateurs de modèle dans une boîte à outils graphique non conventionnelle.

– **Le contrôle dédié :** Ce niveau décrit le contrôle d'une application interactive à travers des *dispositifs d'application*. C'est le niveau de contrôle le plus puissant, car il permet de décrire des techniques d'interaction dédiées à la tâche. Il ne concerne cependant que les applications qui ont été développées pour être compatibles avec ICON, ou des applications qui ont été modifiées dans ce sens. Les dispositifs d'application sont en principe spécifiques à une application donnée, mais il est également possible d'imaginer des librairies de dispositifs réutilisables adaptés à un type particulier d'application. ICONDraw est un exemple d'application qui supporte le contrôle dédié comme le dessin à la tablette graphique (figure 4.24 page 134).

5.2.3 Une gestion des entrées entièrement explicite

Nous avons vu qu'ICON permettait de court-circuiter une bonne partie des services du système d'exploitation et de la boîte à outils afin de gérer la quasi-totalité de l'interaction en entrée. Cette dernière option est la plus avantageuse car elle remplace l'ensemble des boîtes noires standard par une boîte blanche hautement configurable.

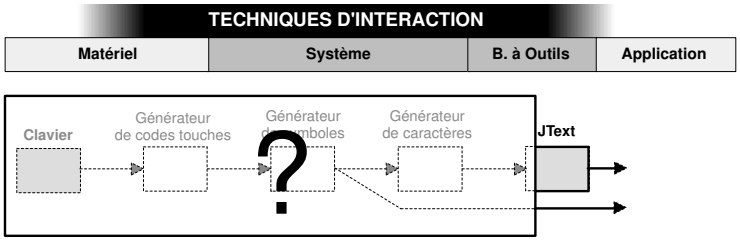


FIG. 5.3 – La cascade de traitements dans un système interactif classique, vue de l'application. Le dispositif concret n'a pas d'existence, et les traitements bas-niveau sont cachés.

La figure 5.3 reprend l'exemple de la gestion standard du clavier initialement introduit dans la section 3.2.5 page 98 [FEKETE & DRAGICEVIC 00]. Certaines fonctionnalités comme la répétition des touches, la mémoire tampon, et la gestion des voyants n'ont pas été mis en évidence dans cette exemple, mais celui-ci reproduit néanmoins les principaux mécanismes de gestion du clavier présents dans tous les systèmes interactifs existants. Comme évoqué précédemment, la gestion des entrées est répartie entre différents acteurs, de l'électronique du dispositif jusqu'à l'application.

Si une grande partie de la gestion des dispositifs d'entrée dans les systèmes interactifs actuels peut être décrite par de telles cascades de dispositifs, cette structuration est toujours *implicite* : les adaptateurs ne sont pas toujours bien distincts, les tâches de traitement sont disséminées à travers différentes couches étanches, et il n'existe finalement pas de mécanisme fournissant une vision globale de la chaîne complète de ces traitements. Bien que les systèmes d'exploitation produisent parfois des données à plusieurs niveaux d'abstraction pour répondre aux besoins des applications et des boîtes à

outils (par exemple, symbole ou code touche en plus du caractère), la partie bas-niveau des traitements relatifs à un dispositif concret est entièrement cachée, ce qui restreint considérablement le champ d'action du développeur (figure 5.3 page ci-contre).

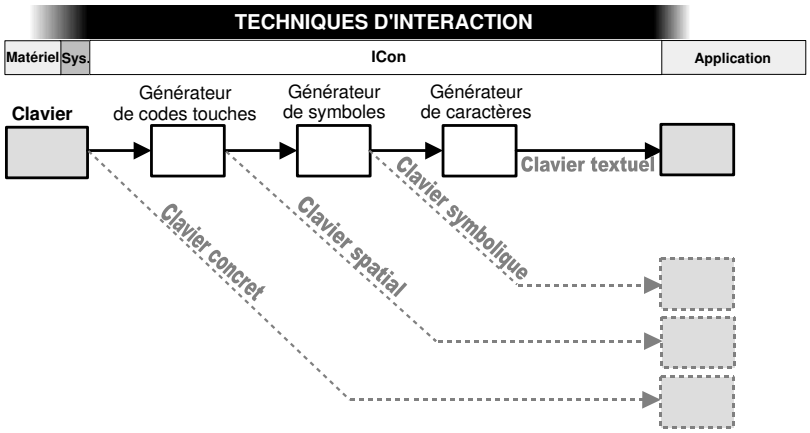


FIG. 5.4 – Les différents niveaux d'abstraction du clavier, explicités par la cascade de dispositifs.

À l'inverse, ICON est capable d'exposer de façon explicite le dispositif physique[1] et tous les traitements effectués en amont. Chez le développeur, la vision synthétique et événementielle d'un dispositif d'entrée est alors remplacée par une vision à niveaux multiples. Le niveau le plus bas est le dispositif concret : le clavier concret est une boîte à boutons, chacun de ces boutons comportant deux états. Cette vision mécanique du clavier est celle qu'expérimente l'utilisateur. Ensuite, chaque adaptateur supplémentaire joue le rôle d'un opérateur logique qui transforme successivement un clavier concret en clavier spatial, puis symbolique, et enfin textuel (figure 5.4). Selon ses besoins, l'application peut se greffer à chacun de ces niveaux : un clavier textuel est pertinent pour une application de traitement de texte, mais un clavier plus bas niveau est mieux adapté à des actions telles que le déplacement d'un curseur (figure 5.4). Il est en outre évident qu'une telle modularité procure une configurabilité bien plus importante.

5.3 Pertinence du modèle pour les interactions Post-WIMP

Nous avons montré dans le premier chapitre l'intérêt des dispositifs d'entrée non standard et des paradigmes d'interaction post-WIMP. Il est par conséquent utile de déterminer dans quelle mesure les outils fournis par ICON et le modèle des dispositifs en cascade sur lequel il repose permettent de décrire ces nouveaux dispositifs et paradigmes d'interaction.

Nous commencerons par évoquer les limitations inhérentes au langage d'ICON et leurs conséquences sur la description de techniques d'interaction en général. Puis, nous nous demanderons si notre modèle permet de prendre en compte l'ensemble des dispositifs non-standard existants et à ve-

[1] Si le plus bas niveau visible est en pratique celui fourni par le pilote de dispositif, des niveaux plus bas peuvent être reconstruits par « *rétro-conception* », comme nous l'avons fait pour les canaux booléens de notre dispositif clavier. Cependant, nous savons déjà que de l'information est perdue au fil des traitements et des abstractions. [ACCOT *et al.* 97] donne un bon aperçu de ce type de problème.

nir. Enfin, nous tenterons de déterminer en quoi notre approche est adaptée ou non à la description des différents paradigmes d'interaction non-standard évoqués dans le premier chapitre.

5.3.1 Limitations du langage et pouvoir d'expression

Le langage à flot de données d'ICON comporte certaines limitations inhérentes. Nous les justifions ici, et décrivons en quoi elles ne nuisent pas réellement à son pouvoir d'expression.

Unicité des connexions en entrée

ICON n'autorise pas les connexions multiples sur un seul slot d'entrée, ce type de connexion n'étant pas souhaitable dans un paradigme réactif. En effet, plusieurs signaux peuvent converger pendant un tick là où l'on suppose qu'un seul signal (et une seule valeur) peut être stocké. La méthode consistant à conserver le dernier signal induit un comportement non-déterministe car l'ordre dans lequel arrivent les signaux n'est pas connu.

Dans ICON, les signaux nécessitent d'être fusionnés explicitement par des dispositifs de type opérateur mathématique ou logique. Cette méthode présente l'avantage d'être déterministe, et permet à l'utilisateur de choisir librement la façon dont il veut combiner plusieurs sources de données sur un seul slot d'entrée.

Staticité

Les configurations d'entrée sont *statiques* : elles ne peuvent pas être modifiées pendant l'exécution. C'est en particulier le cas des connexions. Or les techniques d'interaction comportent à la fois des connexions statiques (la souris bouge toujours le même curseur) et des connexions dynamiques (le curseur contrôle différents objets). En effet, s'il est envisageable d'affecter un dispositif à un objet de façon permanente (par exemple, le contrôle du volume), le nombre de dimensions contrôlables d'une application est souvent supérieur aux dimensions disponibles en entrée, d'où la nécessité d'un *multiplexage* spatial ou temporel.

Le multiplexage peut cependant être décrit avec ICON, de deux manières différentes. La première repose sur des techniques d'*activation sélective* : les sorties d'un dispositif sont connectées de façon permanente aux entrées de plusieurs dispositifs comportant chacun un slot booléen d'activation, et un mécanisme de sélection s'assure qu'un seul d'entre eux est actif à la fois. La configuration montrant un usage de la *Toolglass* dans ICONDraw constitue un exemple d'activation sélective (voir figure 4.34 page 140).

La deuxième technique de multiplexage emploie des *passages de référence* : un dispositif agit sur l'objet qui lui a été transmis en entrée par un autre dispositif, et est capable changer d'objet d'intérêt à tout moment. Les *sélecteurs* et les *manipulateurs* décrits dans la section 4.2.3 page 114 emploient cette technique, et sont notamment utilisés dans la configuration décrivant le contrôle des composants Swing (voir figure 4.18 page 131).

Autoriser des reconnexions dynamiques dans les configurations d'entrée compliquerait significativement le modèle d'exécution, et serait source de confusions importantes si le problème de la représentation visuelle adéquate n'est pas étudié sérieusement. Cependant, si notre approche d'acti-

vation sélective rend explicite l'ensemble des flux de données possibles, elle ne décrit pas clairement pourquoi et quand les données empruntent un chemin plutôt qu'un autre. Nous reviendrons sur ce problème dans la section 5.8.1 page 179 lorsque nous évoquerons les systèmes à transitions.

Acyclisme

Nous distinguons deux types de cycles : les cycles explicites, qui sont internes à la configuration, et les cycles implicites, qui transitent par l'environnement. Nous les décrivons séparément.

Les cycles explicites. En tant que langage réactif, ICON n'autorise pas les *dépendances cycliques directes* : l'algorithme de sa machine réactive repose sur l'existence d'un ordre partiel entre les dispositifs, qui garantit l'exécution de chaque tick en un temps fini. La plupart des langages réactifs résolvent ce problème en autorisant les cycles avec un opérateur de *retard*, qui transmet le signal au tick suivant.

Un opérateur de retard peut également être implémenté dans ICON pour permettre de décrire des *cycles explicites*, c'est-à-dire des cycles internes à la configuration. Cependant nous n'avons jamais eu besoin de tels cycles, sauf à une seule reprise, lorsque nous décrivions un comportement à un niveau de granularité très fin[2]. Les cycles de ce type traduisent des mécanismes internes qui se situent, selon nous, à un niveau trop bas pour être pertinent du point de vue de la personnalisation de l'interaction. Notre expérience a montré qu'à un certain niveau de granularité, les comportements peuvent être tous décrits sans faire appel à des cycles.

Nous pensons plus généralement que les flux *unidirectionnels* de données, où contrôleurs et contrôlés sont clairement séparés, constituent un bon modèle structurant pour l'interaction en entrée bas-niveau. Cette structuration garantit en effet une certaine *amodalité*, dans la mesure où à un niveau donné dans la cascade de dispositifs (niveau qui correspond à une certaine abstraction des dispositifs physiques comme nous l'avons vu dans la section 3.2.5 page 98), l'interprétation des actions de l'utilisateur ne dépend pas de l'état du système en aval.

Les cycles implicites. L'information peut transiter dans n'importe quel sens dans une configuration ICON en passant par les entrées/sorties implicites. Nous qualifions ce type de cycle d'*implicite*, car il transite par l'environnement. Le comportement d'un dispositif *pick*, par exemple, dépend de l'agencement spatial des objets qu'il sert en partie à manipuler. Les cycles implicites sont utiles car ils permettent à une configuration d'entrée d'effectuer, à travers des *points de retour* comme le *pick*, des requêtes dans l'application ou la boîte à outils tout en la contrôlant.

Les seuls cycles qui nous semblent essentiels dans l'interaction sont par conséquent ceux qui résultent de la communication à double sens entre le *contrôle et la vue* d'une part (pour le pick, par exemple), et entre le *contrôle et le modèle* d'autre part (pour le retour sémantique). Ces cycles sont tous implicites et peuvent être décrits par des points de retour. Ils remettent en cause notre modèle unidirectionnel, mais seulement à un niveau de la cascade où la communication bidirectionnelle avec la vue et le modèle devient nécessaire.

[2]Chacun des deux pointeurs décrit par notre technique de curseur vocal (figure 4.33 page 139) peut à tout moment prendre la position de l'autre. Ce mécanisme a été décrit dans ICON dans un dispositif composite (figure 5.8 page 159) en maintenant à jour l'écart entre les deux positions, comportement qui aurait pu être décrit plus simplement par l'usage d'un cycle.

5.3.2 Prise en compte des dispositifs non conventionnels

Les dispositifs d'entrée non standard sont extrêmement nombreux et variés, et de nouveaux dispositifs apparaissent régulièrement sur le marché ou dans le monde de la recherche. Il est par conséquent indispensable de disposer d'un modèle assez général pour prendre en compte les dispositifs existants et ceux à venir.

Le modèle à base de canaux structurés sur lequel repose ICON est à la fois très simple et très général. Les canaux ont été largement employés dans les modèles de dispositifs d'entrée, et sont aujourd'hui adoptés par les principaux standards [USB/HID 01]. En effet, tout dispositif d'entrée peut être décomposé en un ensemble de canaux atomiques. La faiblesse des modèles existants provient non pas de cette décomposition en canaux mais d'une classification figée des dispositifs, qui les rend peu évolutifs et ne permet pas d'exploiter les capacités propres à chaque dispositif. ICON résout le problème en évacuant toute notion de classe *ad-hoc*. Néanmoins, dans certains cas particuliers que nous détaillons ici, notre modèle peut montrer certaines faiblesses.

Les dispositifs à grand nombre de dimensions

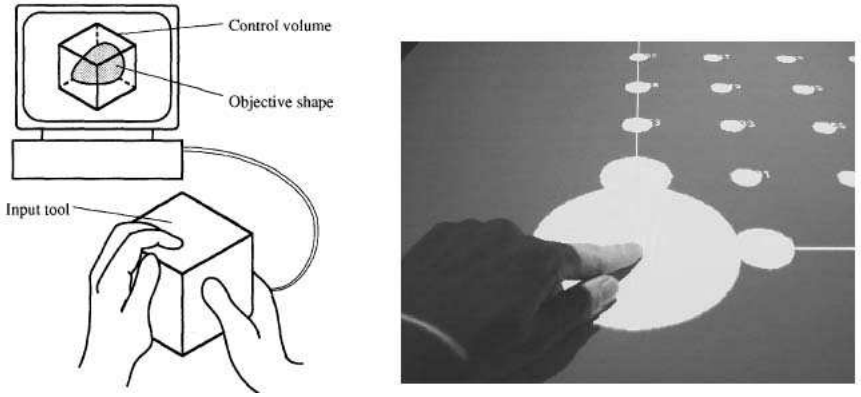


FIG. 5.5 – Deux dispositifs à grand nombre de dimensions : le cube déformable [MURAKAMI *et al.* 95] et la table Smartskin [REKIMOTO 02].

La décomposition en canaux rencontre des difficultés dans les cas de dispositifs d'entrée possédant un très grand nombre de dimensions. Le cube déformable de Murakami et al [MURAKAMI *et al.* 95], représenté sur la figure 5.5 est une structure recouverte de mousse qui comporte 90 degrés de liberté interdépendants. Le dispositif Smartskin de Rekimoto [REKIMOTO 02], déjà présenté au premier chapitre et illustré sur la figure 5.5, image de droite, est une matrice de 72 (ou 768 selon le modèle) capteurs capacitifs. Les nombreux canaux de ces dispositifs seraient pénibles à connecter individuellement. En outre, la représentation en « liste de slots » dans ICON n'est pas adaptée à des canaux disposés en matrice. Même si ICON permet d'organiser ces canaux multiples en un unique slot composite, ceux-ci se comporteraient du point de vue de la machine réactive comme des canaux individuels dont la gestion resterait lourde.

Une solution consiste à modéliser ces dispositifs à un plus haut niveau (par exemple, remplacer la matrice de Smartskin par une liste de pointeurs), avec cependant pour inconvénient de faire disparaître le dispositif concret et de rendre implicite une partie importante de la chaîne de traitements. Pour le

dispositif clavier d'ICON, une solution intermédiaire a été implémentée, où chaque touche (booléen) est accessible individuellement, en même temps que le code (entier) de la dernière touche appuyée ou relâchée. Cette *redondance* est nécessaire, car les deux structures sont pertinentes, selon l'utilisation que l'on désire faire du clavier dans la configuration d'entrée.

Les dispositifs à très haut débit d'informations

Certaines entrées comme les *signaux audio* ou *vidéo* émettent des flux de données extrêmement denses. Ces entrées provoquent une première difficulté d'ordre technique, car le langage Java n'est actuellement pas capable de traiter un tel débit en temps réel, traitement qui est habituellement implémenté au niveau matériel ou dans un langage très bas niveau. Mais surtout, il est difficile de faire cohabiter dans une même machine réactive des algorithmes de traitement de nature totalement différente, car son temps de réaction total est au moins égal à celui du dispositif le plus lent. Or la réactivité n'est qu'une question d'ordre de grandeur : pour la plupart des configurations d'entrée, un temps de réponse comparable au taux de rafraîchissement de l'écran (de l'ordre de 100 Hz) est plus qu'acceptable du point de vue de l'interaction ; pour les applications de reconnaissance gestuelle, la fréquence acceptable est de l'ordre de 1000 Hz, comparable à celle d'une tablette graphique ; pour les applications de traitement de signal audio, la fréquence est de l'ordre de 10000 Hz.

Dans ICON, il est possible de faire communiquer par des entrées/sorties implicites plusieurs configurations d'entrée qui s'exécutent à des vitesses différentes. Cependant, tout comme pour le problème précédent, les dispositifs à très haut débit gagneraient à être modélisés à plus haut niveau, quitte à violer notre principe selon lequel les dispositifs doivent être décrits à un niveau très bas. En effet, dans un système interactif un échantillon audio isolé n'a pas plus de sens qu'un pixel provenant d'une caméra, et il est plus judicieux de transmettre à travers les canaux de ces dispositifs des paquets audio et vidéo. Le dispositif de commande vocale actuellement implémenté dans ICON est un dispositif d'entrée pur, le flux d'informations entre le microphone et le moteur de reconnaissance vocale étant implicite. Les traitements en amont sont par conséquent configurés par paramétrage du dispositif (mots à reconnaître) au lieu d'être représentés explicitement dans ICON.

5.3.3 Description de techniques d'interaction non conventionnelles

À partir d'un nombre limité de dispositifs fournissant les services de base, les possibilités offertes par ICON pour décrire des techniques d'interaction existantes ou inédites sont extrêmement nombreuses. Nous avons construit et testé approximativement une centaine de configurations d'entrée, et nous ne pouvons malheureusement pas toutes les évoquer. Dans le chapitre précédent, nous avons essayé d'en fournir un échantillon représentatif.

Les techniques d'interaction non conventionnelles proposées dans la littérature scientifique sont néanmoins extrêmement nombreuses, et nous n'avons pas pu toutes les décrire à ce jour. Dans cette section, nous exploitons toutefois les résultats obtenus avec les techniques déjà implémentées et tentons de déterminer les capacités potentielles d'ICON pour les autres, dans le but d'identifier les caractéristiques des paradigmes non conventionnels par rapport auxquelles ICON est pertinent et celles pour lesquelles il l'est moins. Nous avons identifié un certain nombre de ces caractéristiques qui nécessitent une prise en charge spécifique :

1. **Le parallélisme :** Le parallélisme est un caractère essentiel des interfaces post-WIMP, en par-

ticulier dans celles basées sur l'*interaction bimanuelle* et *multimodale*. Parce qu'ICON s'appuie sur un paradigme à flot de données, il est naturellement adapté à la description de la concurrence. D'une part, des comportements concurrents peuvent être ajoutés à un comportement existant sans manipuler la configuration d'entrée initiale. D'autre part, le langage graphique se prête bien à la représentation de comportements massivement parallèles, car bien que les connexions soient nombreuses le parallélisme n'induit pas de croisements entre celles-ci. Dans le chapitre sur la boîte à outils ICON, nous décrivons un exemple où un simple copier-coller transforme une interaction Swing standard en une interaction multi-pointeurs (figure 4.22 page 133).

2. **La transparence :** L'affichage d'objets transparents en incrustation est une technique présente dans la plupart des paradigmes post-WIMP, comme les *outils semi-transparents* ou l'*interaction gestuelle*. Les dispositifs de feedback se prêtent bien à la description de telles techniques, et ceux que nous avons implémenté dans ICON nous ont permis de décrire des interactions à base de pointeurs multiples, de barres d'outils et de saisie gestuelle flottantes. Nous pensons plus généralement que ce type de retour graphique permet de décrire un grand nombre de techniques d'interaction indépendamment de l'application, et qu'ICON gagnerait à être étendu pour inclure explicitement un modèle de feedback graphique multicouche tel que celui décrit dans [FEKETE & BEAUDOUIN-LAFON 96, FEKETE 96B].

3. **Les modalités naturelles :** Certaines interfaces post-WIMP reposent sur l'exploitation de modalités naturelles telles que la parole et le geste pour communiquer avec l'ordinateur. Ce type d'entrées, dont l'interprétation est complexe, s'oppose aux entrées explicites telles que le pointage. Pour pouvoir être interprétés par l'application, les signaux bruts provenant de ces entrées doivent être convertis en symboles par des techniques de *classification*. Les algorithmes de classification sont lents et se prêtent mal à un paradigme réactif[3]. Le principe des *entrées/sorties implicites* (voir section C.5.1 page 232) d'ICON permet toutefois de décrire des dispositifs asynchrones et temporellement non-déterministes, qui effectuent leurs calculs dans un autre fil d'exécution afin de ne pas bloquer la machine réactive. Le classificateur est alors vu comme un dispositif d'entrée qui émet des sorties à son rythme propre. Le dispositif de commande vocale (voir section 4.2.1 page 111) en est un exemple. Certains aspects plus haut-niveau de l'interaction multimodale tels que la gestion de l'*ambiguïté* et la *fusion* des modalités ne sont cependant pas pris en charge par ICON.

4. **Les capteurs passifs :** L'interaction implicite à base de capteurs et la sensibilité au contexte sont des caractéristiques prédominantes dans certains paradigmes tels que l'*informatique diffuse* ou l'*informatique vestimentaire*. Les capteurs passifs se distinguent des dispositifs d'entrée en ceci qu'ils ne sont pas directement contrôlés par l'utilisateur. Dans ICON, les capteurs peuvent être tout-à-fait assimilés à des dispositifs d'entrée et être connectés à des applications de la même manière. Nous pourrions par exemple construire une configuration où le niveau acoustique de l'environnement (information déjà disponible à travers le dispositif de commande vocale) agit sur le volume sonore d'une application. L'implémentation d'une grande variété de capteurs ainsi que des dispositifs de traitement adaptés tels que des classificateurs suffirait à décrire la plupart des techniques d'interaction implicite employées dans ces interfaces post-WIMP.

[3] Les interactions à base de classification sont quelque peu en contradiction avec la nature supposée *concise* et *directe* de l'interaction post-WIMP (voir section 1.3.1 page 15). Mais il existe à proprement parler plusieurs approches de l'interaction Post-WIMP, dont certaines s'attachent plus à l'aspect communicationnel qu'à l'aspect instrumental. Quoi qu'il en soit, les interactions à base de classification restent indispensables dans certains contextes comme l'accessibilité.

5.4 Complexité et lisibilité

Un éditeur graphique de configurations possède de nombreux avantages par rapport à des langages de script ou de programmation. Par exemple, dans l'hypothèse où toute notion de classe a été évacuée, les moyens d'accéder à un nouveau dispositif d'entrée ne peuvent pas être rendus explicites dans un langage textuel : il est nécessaire de consulter une documentation ou de faire des requêtes préalables sur le dispositif pour connaître le nom de ses différents canaux. Dans l'éditeur d'ICON, les interfaces des dispositifs sont explicites, et il suffit de connecter les canaux du dispositif d'entrée pour pouvoir l'utiliser.

Cependant, les langages visuels possèdent également de nombreux inconvénients. En particulier, ils supportent mal la complexité, et un programme peut rapidement devenir illisible et difficile à maintenir. ICON n'échappe pas à cette règle (voir par exemple les configurations des figures 5.6 et 4.33 page 139). Dans cette section, nous étudions plus en détail le problème de la complexité visuelle dans ICON.

5.4.1 Le rôle de la structuration spatiale

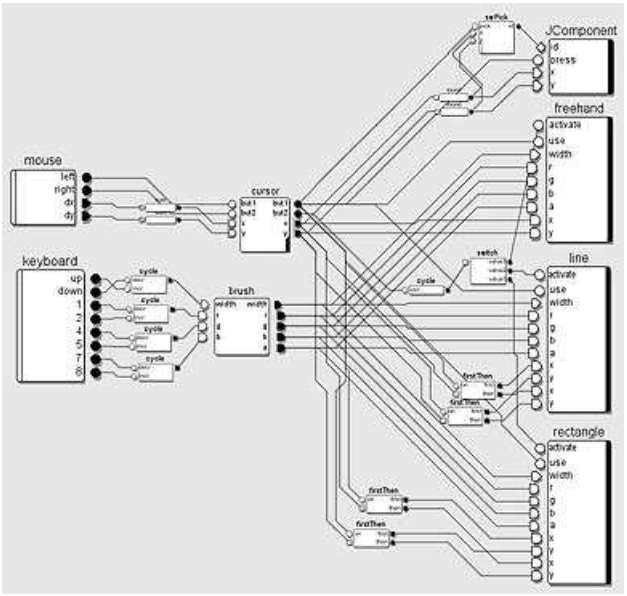


FIG. 5.6 – La configuration par défaut d'ICONDraw, où les dispositifs standard contrôlent différents outils de dessin ainsi que les composants Swing. Cette représentation n'emploie pas de liens.

Sur la figure 5.6 est représentée la configuration d'entrée par défaut d'ICONDraw. C'est une configuration minimale mais complète permettant de contrôler, avec les dispositifs standard, les trois outils de dessin, les attributs de la brosse, ainsi que les widgets standard de Swing. Bien que l'on puisse y deviner une technique de multiplexage, cette configuration est illisible. Non seulement les nombreux croisements entre les connexions (impossible à éviter ici) nuisent à sa lecture, mais il est également difficile de déterminer les rôles respectifs des différents dispositifs.

La figure 5.7 page suivante montre une autre représentation de la même configuration. Cer-

tains dispositifs ont été dupliqués à plusieurs endroits de l'espace de travail en employant des *liens* (voir 4.4.1 page 127). Les dispositifs ont ensuite été regroupés afin de mettre en évidence les différentes parties fonctionnelles de la configuration : la partie *A* décrit le contrôle standard des composants Swing. La partie *B* décrit la sélection de l'outil au bouton droit de la souris, selon une technique de multiplexage temporel. La partie *C* décrit le contrôle des attributs de la brosse au clavier, selon une technique simple d'incrémentation/décrémentation (la brosse est un dispositif « virtuel » fourni par ICONDraw, qui se contente de décrire une structure et transmettre en sortie les données reçues en entrée). Enfin, les parties de D_1 à D_3 décrivent le contrôle des trois outils de dessin.

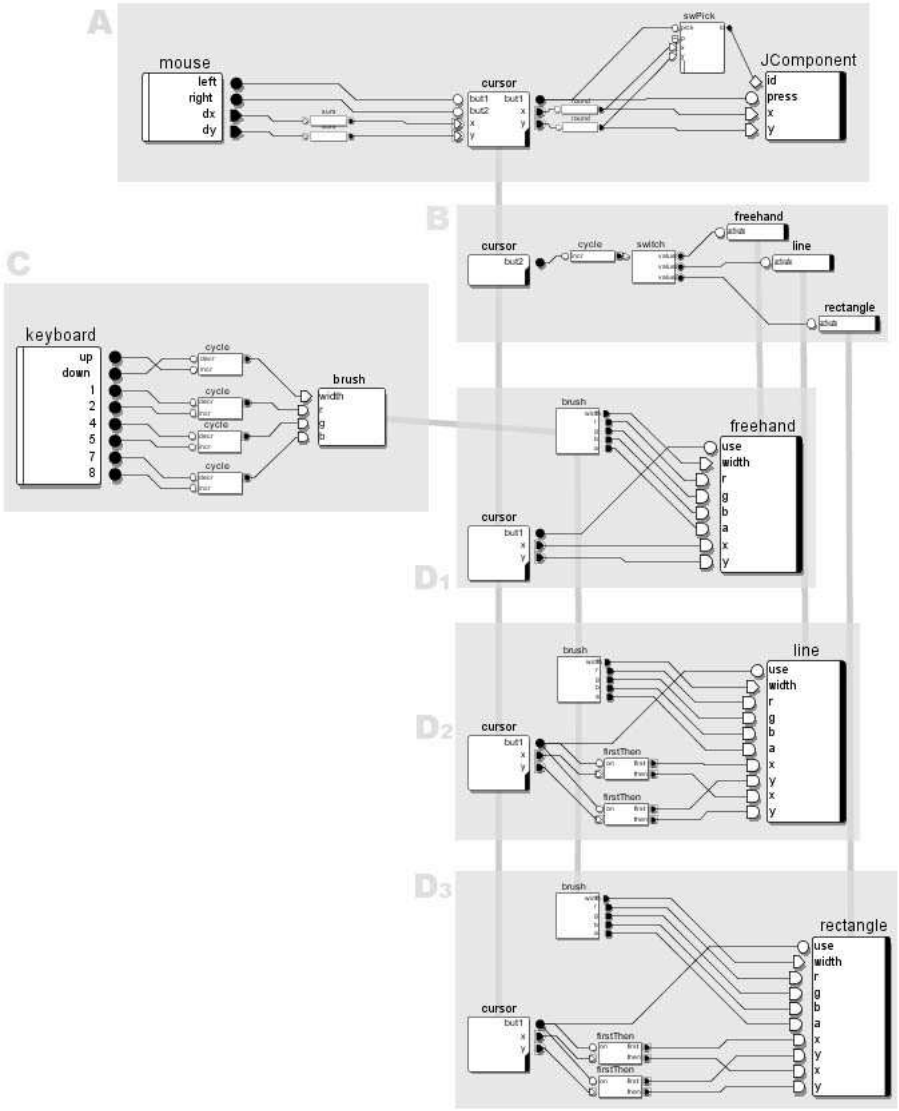


FIG. 5.7 – Sur cette représentation de la configuration par défaut d'ICONDraw, des liens ont été employés pour éviter les croisements de connexions et les dispositifs ont été regroupés de façon à mettre à jour les différentes parties fonctionnelles. Nous avons ajouté les rectangles englobants et les lettres de *A* à *D* pour les besoins de notre discours.

Cet exemple montre que la lisibilité d'une configuration d'entrée repose en grande partie sur une

bonne structuration spatiale. Nous pensons que même des configurations d'entrée extrêmement complexes peuvent être structurées en sous-configurations qui peuvent être comprises et modifiées de façon indépendante. Cette structuration est facilitée dans ICON par les techniques de manipulation directe et par le principe des *liens*, qui sont représentés dans la dimension verticale perpendiculairement aux flots de données. En outre, le *redimensionnement* relatif des dispositifs et l'*encapsulation* de sous-configurations dans des dispositifs composites permettent de mettre en évidence certaines parties de la configuration ou de définir des niveaux de détail. Toutes les techniques n'ont pas été explorées : par exemple, l'*annotation* des configurations à des fins de documentation, ou l'utilisation du *zoom sémantique* pour naviguer dans la hiérarchie des dispositifs composites sont deux techniques qui semblent prometteuses.

5.4.2 Les limites du tout-visuel

Dans le chapitre précédent, nous avons décrit une technique de pointage vocal employant deux pointeurs interdépendants, le premier étant sensible au niveau sonore et le second à la commande reconnue (configuration de la figure 4.33 page 139). Cette technique d'interaction sophistiquée est principalement implémentée par le dispositif *SpeechCursor*, un adaptateur qui renseigne la position des deux pointeurs en fonction des commandes qu'il reçoit.

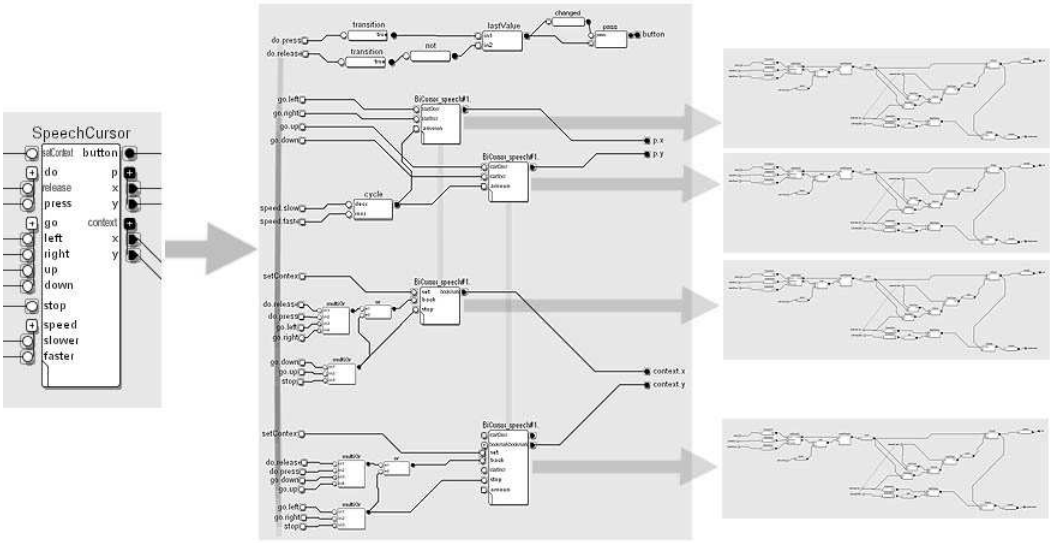


FIG. 5.8 – Décomposition du dispositif composite *SpeechCursor* (à gauche) en sa configuration-fille (au milieu). Cette dernière emploie également quatre instances d'un dispositif composite créé pour l'occasion et dont la configuration-fille est représentée à droite.

Le comportement du dispositif *SpeechCursor* a été entièrement décrit avec des opérateurs élémentaires d'ICON afin de tester la puissance d'expression du langage (figure 5.8). Nous ne nous attarderons par à décrire ces configurations, notre but étant simplement de montrer que malgré les deux niveaux de composition, chaque configuration-fille est relativement complexe et difficile à lire.

Cet exemple montre les limites du « tout-visuel » ou du « tout-modulaire » consistant à décrire n'importe quel comportement, même très complexe, par composition d'opérateurs simples. Bien que

nous ayons à plusieurs reprises insisté sur les avantages apportés par un modèle modulaire en termes de configurabilité, nous avons également montré l'intérêt d'employer des dispositifs monolithiques (implémentés en Java) pour décrire des comportements complexes, et en particulier des techniques d'interaction connues. Nous pensons en effet qu'à partir d'un certain niveau de granularité, il devient inutile de décomposer les comportements car la complexité l'emporte sur la souplesse. En effet, un dispositif monolithique mais paramétrable est toujours plus *configurable* (dans le sens que nous avons défini dans la section 1.4.2 page 29) qu'une configuration d'entrée dont on n'arrive pas à comprendre le fonctionnement. Il nous semble cependant difficile d'établir précisément cette granularité-limite.

5.4.3 Complexité et nature de l'interaction

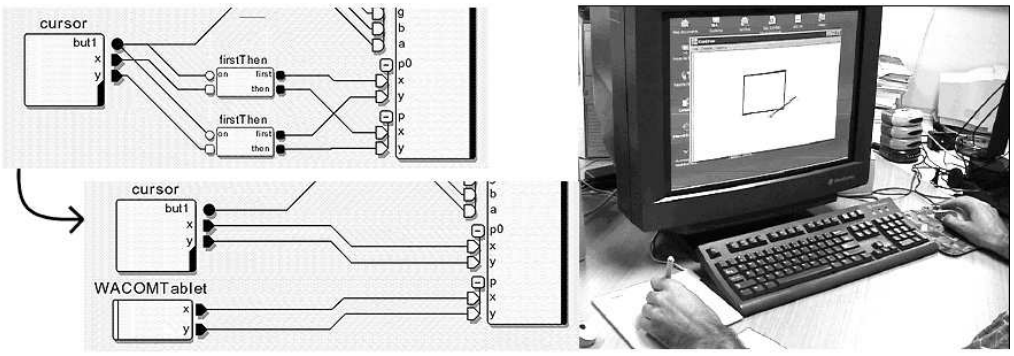


FIG. 5.9 – Tracé de lignes standard et bimanuel avec ICONDraw.

La figure 5.9 illustre deux techniques de tracé de lignes que nous avons déjà évoquées dans le chapitre précédent, et qu'il nous semble utile de reproduire ici. La configuration du haut décrit la technique de *cliquer-glisser* standard couramment employée pour spécifier deux points avec un seul pointeur. Celle-ci a été ensuite modifiée (configuration du bas) pour prendre en compte un deuxième dispositif de pointage. La configuration obtenue est bien plus simple que la configuration originale, d'abord parce que le multiplexage n'est plus nécessaire : deux dispositifs contrôlent deux dimensions. Ensuite, parce que la tablette graphique fournit des données directement compatibles avec celles du dispositif d'application (la souris nécessite la conversion de positions relatives en positions continues).

Cet exemple montre que *la complexité d'une configuration d'entrée dépend d'une certaine manière du type d'interaction qu'elle décrit*. Les techniques d'interaction standard, parce qu'elles utilisent des entrées relativement *pauvres* et *génériques*, emploient des techniques d'interaction essentiellement basées sur le *multiplexage*, nécessitant l'usage de modes et de structures conditionnelles. Ces comportements peuvent être à l'origine de configurations d'entrée relativement complexes. À l'inverse, les configurations d'entrée les plus simples, où la transformation des données est majoritaire par rapport à leur contrôle, décrivent des techniques de *contrôle direct* avec des *dispositifs multiples*. La technique d'interaction bimanuelle décrite ci-dessus en est un exemple, tout comme le contrôle d'une interface zoomable avec un dispositif à six degrés de liberté (figure 4.25 page 134).

Il est intéressant de constater que dans l'exemple décrit précédemment, la technique d'interaction considérée comme la plus naturelle se décrit aussi le plus aisément. Une technique d'interaction simple d'utilisation ne se décrit pas nécessairement de façon simple dans ICON (par exemple, un dispositif

d'entrée et un objet de l'application peuvent être compatibles du point de vue cognitif mais être modélisés différemment). En outre, la simplicité d'une configuration d'entrée ne garantit rien, bien sûr, quant à l'utilisabilité de la technique qu'elle décrit (la suppression aléatoire d'une connexion simplifie la configuration mais pas l'interaction). Malgré tout, nous pensons qu'il existe un rapport entre la simplicité d'une configuration d'entrée et certaines « bonnes » propriétés de l'interaction, en particulier celles qui caractérisent les paradigmes post-WIMP (voir section 1.3.1 page 15).

En effet, nous pensons tout d'abord qu'un critère d'utilisabilité essentiel pour une technique d'interaction de type instrumental[4] est sa faculté d'être *facilement compréhensible* par l'utilisateur : celui-ci doit pouvoir l'assimiler facilement mais surtout ensuite pouvoir prédire son comportement, et pour ce faire, s'être construit un modèle mental précis de son fonctionnement. En conséquence, les mécanismes de cette interaction doivent pouvoir se décrire facilement dans le formalisme qui convient. Nous pensons par ailleurs que les langages à flots de données, et ICON en particulier, sont bien adaptés à la description de la plupart des interactions de type Post-WIMP, car ils encouragent la description de techniques d'interaction *directes*, *amodales* et *parallèles*.

5.5 Caractérisation des différents types de dispositifs

ICON permet de spécifier des techniques d'interaction en termes de configurations d'entrée, mais ne décrit pas précisément comment ces configurations doivent être décomposées en dispositifs, ni la façon dont ces dispositifs doivent être décomposés en slots. Nous avons en partie répondu à cette question dans le chapitre précédent par de nombreux exemples de dispositifs et de configurations. Dans ICON, la plupart des choix de conception restent cependant à la charge du développeur.

Le *dispositif* est la brique de base dans ICON, et il est essentiel de se demander ce qui le caractérise et ce qu'il représente. Du point de vue des entrées, la plupart des dispositifs physiques sont des objets autonomes dont les parties constituantes sont indissociables : faire correspondre à un dispositif physique un dispositif dans ICON est par conséquent une bonne approximation. La décomposition d'une application en dispositifs d'application est cependant moins triviale, et les choix relatifs à cette décomposition incombent au programmeur d'application. La décomposition d'une configuration d'entrée en techniques d'interaction pose également la question du bon niveau de granularité.

Dans cette section, nous définissons plus précisément les rôles et les fonctions des dispositifs système d'entrée, des dispositifs de boîte à outils et d'application, et des dispositifs utilitaires. Nous décrivons notamment les services que ces dispositifs, et en particulier les dispositifs d'application et de boîte à outils, sont sensés fournir. Nous énumérons également les difficultés que le programmeur peut rencontrer pour décrire correctement ces dispositifs, et proposons des directives d'implémentation permettant de guider le développement de nouveaux dispositifs.

[4] C'est-à-dire que l'interface est manipulée comme un instrument ou un outil plutôt qu'employée comme un médium de communication.

5.5.1 Les dispositifs système d'entrée

Rôle et fonction des dispositifs système d'entrée

Un dispositif d'entrée ICON est un dispositif système qui décrit un périphérique utilisateur *physique*. En pratique, le rôle des dispositifs d'entrée dans ICON est de fournir une vue *unifiée* et *bas-niveau* des dispositifs décrits par les diverses *APIs d'entrée*.

L'implémentation de dispositifs système d'entrée peut poser un certain nombre de difficultés, que nous décrivons par la suite.

L'hétérogénéité des APIs d'entrée

Le modèle de chaque dispositif système (structure et nom des slots, en particulier) sera intimement lié à l'API sur lequel il repose, et sur les choix qui y auront été faits. Les APIs existantes étant assez hétérogènes, deux dispositifs système représentant le même dispositif physique mais provenant d'APIs différentes pourront ainsi avoir une structure toute différente. En particulier, il n'existe pas de convention de nommage pour les slots.

L'hétérogénéité des APIs n'est cependant pas limitative dans le sens où ICON ne nécessite ni abstraction ni sémantique dans son modèle de configurations. Un utilisateur d'ICON a simplement besoin de voir apparaître les dispositifs concrets dont il dispose, et d'être capable d'identifier ces dispositifs et la sémantique de leurs canaux. Cette identification repose uniquement sur le choix (en général judicieux) des noms (nom du dispositif, nom de ses canaux) faits par les fabricants des dispositifs et de pilotes.

Le problème de la réutilisabilité des configurations d'entrée d'un système à l'autre sera abordé plus loin dans cette discussion (section 5.6.3 page 171). Enfin, notons que le problème de l'hétérogénéité pourrait être en partie résolu par l'emploi exclusif d'une API « universelle » telle que le USB Human Interface Device [USB/HID 01].

Le choix du bon niveau d'abstraction

Il existe à proprement parler au moins trois niveaux « bas » : le bas-niveau logique (données numériques en provenance du pilote de dispositif), le bas-niveau électronique (les signaux électroniques) et le bas-niveau physique (ou mécanique). Ce dernier est le plus pertinent, car il est en contact direct avec l'utilisateur, et contribue le plus à l'image mentale que celui-ci se fait du dispositif et de ses actions sur celui-ci. Or les APIs d'entrée se situent au mieux au niveau logique, c'est-à-dire au plus bas niveau logiciel. Et il n'existe pas nécessairement de correspondance directe entre ce niveau logique et le modèle physique du dispositif.

Non seulement une modélisation très bas-niveau n'est pas toujours réalisable, mais elle n'est pas non plus adaptée à tous les contextes d'utilisation. À titre d'exemple, un clavier devra apparaître comme un ensemble de slots booléens (un slot par touche) pour être fidèle à son modèle physique. Cette structure est pertinente lorsque l'on doit s'intéresser à un ensemble restreint de touches (touches fléchées, par exemple). Mais elle devient trop lourde lorsque l'on doit appliquer un filtre à l'ensemble du clavier, car elle nécessite de connecter chaque touche parmi les centaines existantes. L'utilisation de données sérialisées à base de codes touches peut s'avérer nécessaire.

Ces considérations nous amènent à définir trois propriétés souhaitables pour les dispositifs d'entrée, propriétés qui doivent guider les programmeurs :

Abstraction physique : Les slots qui décrivent le dispositif doivent être choisis et structurés en fonction des caractéristiques physiques de ce dispositif. Les données qui proviennent de ces canaux doivent être brutes, ou subir des transformations sans perte si elles ne coïncident pas avec le modèle physique du dispositif.

Redondance : Si le besoin s'en fait sentir, il peut être nécessaire de fournir des manières alternatives d'accéder aux données du dispositif, par des slots supplémentaires de plus haut niveau (valeurs absolues en plus de relatives, informations synthétiques sur plusieurs canaux, etc.).

Paramétrisation : Si le besoin s'en fait sentir, il peut également être nécessaire de définir des comportements alternatifs pour un même dispositif. Les traitements de données qui implémentent ces comportements seront bien identifiés et choisis par l'utilisateur dans les *paramètres* du dispositif.

5.5.2 Les dispositifs d'application et de boîte à outils

Rôles et fonctions de ces dispositifs

Le rôle principal des dispositifs d'application et de boîte à outils est de fournir des ***points d'entrée*** : pour les premiers, il s'agira d'exposer une interface de contrôle pour une application donnée et de prendre en charge ce contrôle. Pour les seconds, il s'agira d'exposer une interface de contrôle générique pour l'ensemble des applications qui reposent sur la même boîte à outils graphique.

Une autre fonction de ces dispositifs est de fournir des ***points de retour***. Ces derniers sont concrétisés par des dispositifs comportant des entrées implicites et produisant des sorties, et qui peuvent être employés pour effectuer des requêtes vers l'application ou la boîte à outils. Les *sélecteurs* de Swing comme le pick, décrits dans le chapitre précédent, sont des exemples de tels dispositifs.

Dans la suite, nous décrivons un ensemble minimal de services utiles que pourraient fournir à ICON les applications et les boîtes à outils, sous forme de points d'entrée et de retour.

Services communs aux applications

Jean-Daniel Fekete [FEKETE 96A] a identifié trois services du noyau sémantique indispensables à l'interaction : la *notification*, la *prévention des erreurs*, et l'*annulation*. Ces trois services, s'ils sont pris en charge par l'application, peuvent être externalisés sous forme de points d'entrée ou de points de retour :

- **La notification** est principalement employée pour mettre à jour la vue, mais pourrait également se révéler utile comme point de retour dans une configuration d'entrée. Un tel dispositif de retour peut par exemple comporter en sortie un booléen mis à vrai à chaque changement, ou un slot produisant des listes d'objets ayant changé.
- **Le retour sémantique** est souvent employé dans l'interaction en entrée pour prévenir les erreurs. Le retour sémantique peut être décrit par des points de retour qui déterminent si une opération est valide ou non. Chaque dispositif peut représenter une commande, ou recevoir la commande dynamiquement en entrée. Les arguments de la commande sont des objets transmis sur les slots d'entrée. En sortie des dispositifs, un booléen indique si la commande est valide, et

une chaîne peut éventuellement transmettre un message d'erreur. Un filtre produisant une liste d'arguments valides serait également utile pour décrire des techniques comme le *snap sémantique* [HUDSON 90] ou le *drag-and-pop* [BAUDISCH *et al.* 03].
– **L'annulation** est un point d'entrée qui peut être présent sur chaque dispositif sous forme d'un slot *annuler* et d'un slot *refaire*. L'annulation peut également être décrite dans un unique dispositif global.

Services communs aux boîtes à outils

La *notification* et l'*annulation* sont deux services qui peuvent également être fournis à travers les boîtes à outils graphiques. Voici les autres services de boîtes à outils graphiques susceptibles d'être employés dans les configurations :

– **Points d'entrée :**
  – **Les manipulations génériques :** Il s'agit de points d'entrée qui décrivent l'ensemble de manipulations qu'il est possible d'effectuer sur tout type de widget. Le *manipulateur de surface* DJComponent, qui interprète des manipulations positionnelles simples, en constitue un exemple. Des manipulations géométriques plus sophistiquées peuvent être prises en charge, comme les changements d'échelle et les rotations de Jazz. Un protocole d'émission de commandes similaire à ceux des boîtes à outils semi-transparents (section 2.6.2 page 74) peut également être défini. Enfin, les boîtes à outils peuvent éventuellement donner les moyens de modifier l'emplacement et la hiérarchie de leurs widgets.
  – **Les manipulations spécifiques :** Il s'agit de points d'entrée qui décrivent le contrôle de widgets de types spécifiques. Les *manipulateurs de modèle* DJScrollbar et DJText sont des exemples de points d'entrée qui permettent de décrire des techniques d'interaction adaptées aux modèles internes des widgets.
  – **Le feedback :** Certains feedbacks tels que la mise en évidence graphique peuvent être pris en charge au sein des widgets. Ce service n'est cependant pas indispensable et peut avantageusement être remplacé par la création de fantômes (voir plus bas).
– **Points de retour :**
  – **Le pick :** Il s'agit d'un service indispensable, dont la version minimale consiste à retourner l'objet le plus profond dans la hiérarchie qui se trouve sous une position donnée. Des services plus avancés peuvent consister à prendre en compte des critères de sélection et/ou retourner une liste d'objets candidats. Le pick avec des formes quelconques permettrait également de décrire des techniques positionnelles évoluées comme le pointeur pour main non-dominante de *Bimanual Whizz* (section 2.6.2 page 70) ou la manipulation groupée d'objets.
  – **Le focus :** Si la boîte à outils gère le focus, elle peut indiquer le focus courant par un point de retour et/ou le rendre contrôlable par un point d'entrée. Ce service n'est pas indispensable et pourrait décrit dans ICON à partir des deux services suivants.
  – **Les fantômes :** Ces points de retour produisent des instantanés des objets de la boîte à outils, qui peuvent être animés en incrustation avec ICON. La création d'objets fantômes multiplie les possibilités pour décrire le feedback dans ICON, en particulier si cette fonctionnalité est paramétrable (choix entre contours seuls / forme pleine / image, et choix de la couleur et de la transparence). Les contours d'un objet peuvent également être employés dans des calculs de collision pour des techniques de manipulation à base de formes (alignements [RAISAMO & RÄIHÄ 96] et manipulation groupée [REKIMOTO 02], par exemple).
  – **Informations non graphiques :** Des informations non graphiques (analogues à l'API d'ac-

cessibilité de Swing) peuvent être fournies telles que : le nom unique de chaque widget, sa description, l'ordre de parcours des widgets, et leur structure hiérarchique. Ces points de retour autorisent la navigation non-positionnelle dans l'interface (gestion des cycles de focus, contrôle vocal).

Identifier et décrire les points d'entrée d'une application

Des dispositifs d'application peuvent être développés pour tout type d'application interactive. Il n'existe pas de modèle générique pour ces dispositifs, chaque développeur devant déterminer quel modèle est pertinent pour son application spécifique. Bien que des modèles génériques adaptés à des types particuliers d'application (dessin, traitement de texte, modélisation 3D) puissent être susceptibles de guider les développeurs dans l'implémentation des dispositifs d'application, il est bon de rester à l'écart des modèles simplificateurs et garder l'accent sur l'aspect hautement spécifique des tâches propres à chaque application interactive.

Les objets d'application ne sont pas toujours aussi bien délimités que les dispositifs physiques. En particulier, les développeurs peuvent hésiter devant le niveau de granularité à adopter. Ainsi, chaque commande d'une application (les commandes de menus, par exemple) peut être individuellement exposée sous forme de dispositif, ou toutes ces commandes peuvent être regroupées dans un dispositif plus grand. En outre, les programmeurs peuvent choisir ou non d'exposer certains objets ou mécanismes, selon le degré de configurabilité qu'ils désirent atteindre, et l'effort qu'ils comptent y consacrer.

Nous avons déjà décrit dans le chapitre précédent une taxonomie orientée-objet des dispositifs d'application, utile pour guider les choix : les *dispositifs de classe*, les *dispositifs d'instance statiques* et les *dispositifs d'instance dynamiques* (voir section 4.2.4 page 118). Des exemples de dispositifs d'instance dynamiques sont les outils de dessin d'ICONDraw : chaque dispositif employé dans la configuration crée une instance dans l'application. Les *manipulateurs* de Swing, en revanche, sont des dispositifs de classe, qui opèrent sur des instances variables de la même classe.

Une fois les points d'entrée identifiés, il convient de les décrire correctement. Cette opération comporte des difficultés similaires à celles que nous avons déjà évoqué pour les dispositifs d'entrée. Nous avons par exemple constaté, chez les débutants en développement ICON, des difficultés à appréhender la notion d'« indépendance par rapport aux entrées », difficultés qui ont été à la source d'erreurs de conception. Pour cette raison, il n'est pas inutile de guider les développeurs en leur fournissant, comme nous l'avons fait avec pour dispositifs d'entrée, des directives d'implémentation une fois que les objets à exposer ont été clairement identifiés :

Abstraction applicative : Les slots qui décrivent le dispositif doivent être choisis et structurés en fonction des caractéristiques de l'objet applicatif et uniquement celui-ci. L'objectif qu'il faut garder à l'esprit est l'indépendance par rapport aux entrées : *le dispositif doit être construit sans préjugé sur la façon dont il sera contrôlé*.

Redondance et paramétrisation : En fonction des utilisations envisagées, et tout comme avec les dispositifs système, il peut s'avérer nécessaire de fournir plusieurs manières de contrôler un dispositif d'application ou de le rendre paramétrable. Une valeur numérique pourra par exemple être contrôlée soit par affectation, soit par incrémentation/décrémentation.

5.5.3 Les dispositifs utilitaires

Rôle des dispositifs utilitaires

Le rôle des dispositifs utilitaires est de permettre de connecter à travers des *transformations de données* et du *feedback* un ensemble de dispositifs d'entrée à un ensemble de dispositifs d'application. Ces dispositifs sont à la fois des *adaptateurs* et des *techniques d'interaction*, et peuvent être composés pour former des adaptateurs et des techniques d'interaction plus complexes.

Le choix du bon niveau de granularité

C'est au niveau des dispositifs utilitaires que le problème de la décomposition est le moins trivial. Même si les dispositifs d'ICON sont composables, et qu'il est donc possible de décrire des comportements sur plusieurs niveaux d'abstraction, il convient de choisir jusqu'où s'arrête cette décomposition. Autrement dit, quelle doit être la *granularité* des dispositifs atomiques ?

Bien que séduisante, la démarche consistant à fournir un ensemble de dispositifs de fonctionnalité minimale qui soit suffisant pour décrire n'importe quelle technique d'interaction est critiquable du point de vue pratique : d'abord, il n'est pas évident de déterminer *à priori* cet ensemble minimal. Ensuite, la description graphique de comportements complexes peut se révéler rapidement pénible, comme nous l'avons vu dans notre discussion sur la complexité visuelle.

Aussi, nous pensons qu'il est essentiel de pouvoir disposer d'une bibliothèque de dispositifs mathématiques et logiques simples, mais également d'une bibliothèque évolutive d'*adaptateurs monolithiques* de complexité variable. L'utilisateur a ainsi la possibilité de manipuler et d'implémenter des techniques d'interaction à part entière, de composer des adaptateurs de complexité moyenne, et d'utiliser des dispositifs minimaux là où les outils existants font défaut. Nous avons déjà donné des exemples de ces différentes approches dans la section 4.4.4 page 136.

5.6 Développer avec ICON

Dans cette section, nous analysons sous un angle pragmatique la pertinence de notre approche du point de vue du développeur d'applications. Nous en avons déjà abordé un aspect en faisant état des difficultés relatives aux choix de conception. Ici, nous commençons par déterminer en quoi ICON induit une nouvelle façon de programmer l'interaction, et énumérons les concepts auxquels doivent s'habituer les programmeurs ayant jusque-là employé des outils conventionnels. Puis, nous décrivons dans quelle mesure et avec quelle facilité ICON peut être employé avec des boîtes à outils graphiques et des applications existantes. D'autres considérations pratiques sont ensuite abordées, à savoir la réutilisabilité des configurations, la prise en charge effective des dispositifs d'entrée physiques, et enfin les questions de performance.

5.6.1 Maîtriser un nouveau paradigme de programmation

L'implémentation de dispositifs ICON est relativement simple en soi, car comme nous l'avons vu dans la section 4.3.1 page 118, elle consiste essentiellement à déclarer des slots puis à spécifier les

actions à effectuer lorsque la valeur d'un slot d'entrée a changé (ou plus exactement, lorsque celui-ci a reçu a signal). Il peut exister néanmoins des difficultés qui découlent d'un style de programmation différent des outils habituels. Nous énumérons ici les principaux paradigmes et concepts pouvant poser problème, en évoquant notamment les erreurs-type déduites de l'observation et du suivi de deux programmeurs ayant employé ICON pour développer une application (voir section 4.5 page 141). Nous commençons par un concept essentiel dans ICON, à savoir *l'indépendance par rapport aux entrées*. Puis, nous évoquons trois notions « exotiques » découlant de notre modèle d'exécution, à savoir la *réactivité*, la *simultanéité* et le *déterminisme*. Enfin, nous décrivons les problèmes liés à l'utilisation d'*objets* dans les configurations d'entrée.

L'indépendance par rapport aux entrées : Nous avons déjà constaté dans une discussion précédente que les développeurs habitués aux boîtes à outils conventionnelles avaient quelques difficultés à maîtriser la philosophie d'« indépendance par rapport aux entrées ». Dans ICON, cette philosophie impose que *les dispositifs soient construits sans préjugé sur la façon dont ils seront contrôlés*. Or, il est courant de constater des erreurs consistant à déclarer des dispositifs d'application qui comportent, à titre d'exemple, des slots d'entrée nommés `shift` ou `mouseClick`. Une autre erreur typique consiste à implémenter dans l'application, ou dans le dispositif d'application, des mécanismes qui sont en réalité des techniques d'interaction, et qu'il est nettement préférable de décrire dans la configuration d'entrée. Par exemple, un dispositif d'application est déclaré avec un slot `sélectionner`, qui sélectionne un dossier lorsque sa valeur passe de `faux` à `vrai` ou alors ouvre celui-ci lorsqu'il passe deux fois de `faux` à `vrai` dans un intervalle de temps assez court. Il s'agit d'un comportement modal de type *double-clic*, qui constitue une technique d'interaction et doit être décrit à l'extérieur du dispositif d'application. Si ce type d'erreur est facile à comprendre, la limite entre ce qui doit être décrit dans ICON et ce qui doit être décrit dans l'application n'est pas toujours triviale.

La réactivité : La programmation réactive est un paradigme peu connu des programmeurs utilisant des langages impératifs, et certains de ses concepts comme le temps de propagation nul peuvent leur paraître difficiles à appréhender. Cependant, ni l'utilisateur de l'éditeur interactif ni le programmeur n'ont à se soucier de ces concepts, car ICON gère intégralement et de façon transparente la propagation de l'information entre les dispositifs. Le modèle réactif impose cependant des contraintes sur le temps d'exécution de chaque dispositif (méthode `update()`). Des erreurs courantes consistent à effectuer, dans cette méthode, des initialisations lourdes au lieu de les placer dans la méthode `open()`, à effectuer des opérations qui réclament du temps (gros calculs ou routines graphiques) au lieu de les lancer dans un fil d'exécution séparé, ou à créer des objets à chaque tick là où ce n'est pas indispensable. Dans ce dernier cas, et bien que la distinction puisse être subtile, il convient de différentier les créations *discontinues* d'objets (par exemple, créer un objet à chaque clic) des créations *continues* (par exemple, créer un objet à chaque mouvement de la souris) qui alourdissent l'exécution et occupent rapidement la mémoire.

La simultanéité et la gestion des états : Une autre caractéristique d'ICON découlant de son modèle réactif est la notion de simultanéité. La simultanéité est absente de la programmation événementielle classique, où les événements sont traités séquentiellement en fonction de leur ordre d'arrivée dans la file. Dans le modèle réactif d'ICON, il est fréquent que plusieurs signaux arrivent au même moment dans différents slots d'un dispositif d'entrée (ils peuvent également arriver dans le désordre). L'ordre dans lequel ces signaux sont traités a alors son importance. Par exemple, le dispositif de dessin à main levé d'ICONDraw (voir section 4.2.4 page 117) comporte un mode « utilisé » dans lequel il dessine et un mode « non utilisé » dans lequel il anime simplement une brosse. La position de la brosse est contrôlée par les slots `x` et `y`, et le changement de mode est déterminé par le slot `use`. Pour

ce dispositif, il convient de vérifier si le slot `use` a reçu un signal *avant* de traiter les valeurs `x` et `y`. La simultanéité complique généralement la gestion des modes et des états, car diverses combinaisons possibles de signaux doivent être prises en compte, y compris les combinaisons contradictoires (par exemple, un signal `vrai` est reçu simultanément sur des slots `on` et `off`). En outre, les choix qui sont faits n'apparaissent pas à l'extérieur du dispositif, si ce n'est à travers sa documentation. C'est pourquoi nous pensons que la programmation de certains dispositifs dans ICON gagnerait à reposer sur des outils adaptés, idée qui sera développée dans la section 5.8.1 page 179.

Déterminisme et non-déterminisme : ICON fait la distinction entre *déterminisme* et *non-déterminisme*, autres concepts auxquels un développeur n'est pas nécessairement habitué. Les dispositifs non-déterministes sont simplement des dispositifs qui emploient des informations en provenance de l'*environnement*, système qui englobe tout ce qui ne fait pas partie de la configuration d'entrée. Nous disons dans ICON qu'ils comportent des *entrées implicites* (voir section 3.4 page 101). Par ailleurs, nous disons d'un dispositif qui modifie l'environnement qu'il comporte des *sorties implicites*. ICON ne déduit pas automatiquement la présence d'entrées/sorties implicites, et le développeur du dispositif doit explicitement les déclarer en dérivant les deux accesseurs booléens correspondants. Cette opération, bien que simple, peut être facilement omise par le programmeur. Elle est pourtant indispensable car premièrement, un dispositif non-déterministe est traité différemment par la machine réactive : celui-ci est activé à chaque tick afin qu'il puisse, même en l'absence de signaux d'entrée, surveiller l'évolution de l'environnement et éventuellement transmettre des valeurs en sortie. Ensuite, la présence d'entrées et de sorties implicites est une information qui contribue à une bonne compréhension d'une configuration d'entrée, car elle permet de voir clairement à quels endroits cette dernière communique avec l'extérieur (utilisateur et application notamment).

Manipuler des objets dans les configurations d'entrée : Une configuration d'entrée opère sur des types primitifs, structurés ou non, tels que des entiers ou des booléens. La manipulation d'objets y est également possible grâce aux slots de type `object`, mais celle-ci requiert toutefois certaines précautions. Il existe trois principales manières d'employer des objets dans les configurations d'entrée : la transmission de références, la création d'objets et la manipulation d'objets. La *transmission de références* consiste à transmettre d'un dispositif à l'autre des références à des objets qui proviennent initialement de l'environnement. Les dispositifs sélecteurs/manipulateurs, illustrés notamment par la technique du pick, constituent un exemple de cette utilisation (voir section 4.2.3 page 114). Cet emploi ne pose aucun problème. La *création d'objets* consiste à créer des objets au sein d'un dispositif avant de les passer par référence. La création d'objets ne pose pas de problème tant qu'elle ne nuit pas à l'hypothèse réactive (voir la discussion précédente). Plus problématique est la *manipulation d'objets* qui consiste, au sein des dispositifs, à lire ou à écrire dans les objets transmis. Ces opérations peuvent en effet engendrer des dépendances cachées entre les dispositifs et rendre une configuration d'entrée implicitement non-déterministe. C'est le cas par exemple de la configuration de la figure 5.10 page ci-contre. Lorsque c'est possible, il est toujours préférable de composer des slots primitifs plutôt que d'employer des objets, car les flux d'information et la structure des données manipulées y sont explicites. Dans le cas contraire, tout objet lu ou modifié doit être considéré comme *faisant partie de l'environnement*, et les dispositifs concernés déclarés comme ayant des entrées et/ou des sorties implicites.

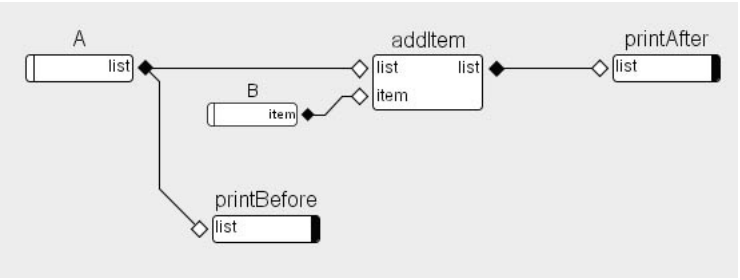


FIG. 5.10 – Configuration d'entrée manipulant des objets de type *Liste* : le dispositif `addItem` concatène les objets émis par `B` à la dernière liste émise par `A`, que `printAfter` affiche après chaque modification. `printBefore` affiche quant à lui les listes émises par `A`. Précisons que seules des références sont transmises dans les connexions. Supposons que `A` émette une référence vers une liste $L = \{a_1, a_2\}$ et qu'au même moment, `B` émette une référence vers un objet a_3. Le dispositif `printBefore` affichera $\{a_1, a_2\}$ ou $\{a_1, a_2, a_3\}$ selon qu'il s'exécute avant ou après `addItem`. Or, comme il n'existe aucune dépendance explicite entre ces deux dispositifs, il n'y a aucun moyen de savoir lequel sera exécuté avant l'autre. Le comportement de `printBefore`, non-déterministe, dépend de la manière dont le tri topologique est effectué.

5.6.2 Connexion avec les outils et applications existantes

Indépendance d'ICON par rapport aux boîtes à outils graphiques

Les boîtes à outils graphiques gèrent les entrées sans les séparer clairement de l'aspect graphique. À l'inverse, ICON ne s'occupe que du premier aspect, et laisse le second aux boîtes à outils graphiques. Il peut par conséquent fonctionner avec plusieurs d'entre elles. Pour l'heure, ICON fournit les dispositifs minimaux pour l'interaction avec les applications Swing, ainsi qu'une prise en charge expérimentale des applications Jazz. À l'avenir, la bibliothèque de dispositifs pourra être enrichie d'autres dispositifs de boîte à outils. Rappelons cependant qu'une application peut être contrôlée indépendamment de la boîte à outils graphique sur laquelle elle repose, avec des dispositifs d'application décrits par le programmeur.

Des applications non-java peuvent également être contrôlées avec ICON. Plusieurs stratégies sont possibles : une interface de communication externe peut être définie dans l'application (un protocole réseau, par exemple), et des dispositifs ICON qui implémentent cette interface peuvent être ajoutés à la bibliothèque. Une application interactive existante peut également être contrôlée sans modification de son code, lorsque celle-ci implémente déjà un protocole bien défini. Nous avons ainsi pu contrôler l'application Microsoft Word (zoom du document, commandes telles que le copier/coller) en implémentant un dispositif qui s'appuie sur le protocole COM (Component Object Model).

Ce type de contrôle a cependant pour inconvénient la perte du feedback graphique et du court-circuitage de la gestion événementielle standard, car dans la version actuelle d'ICON ces fonctionnalités sont implémentées par des dispositifs de la boîte à outils Swing. Néanmoins, ces deux aspects sont conceptuellement indépendants de la boîte à outils. Un feedback graphique peut être implémenté au niveau système, à condition que celui-ci fournisse les services graphiques nécessaires comme l'incrustation et la transparence. De même, dans la mesure où le système d'exploitation le permet, les événements standard peuvent être interceptés à leur source.

Gestion de la concurrence dans les boîtes à outils et les applications

Le problème principal posé par la connexion d'ICON avec des boîtes à outils graphiques ou des applications existantes est qu'elle requiert une coopération minimale de ces dernières : leur contrôlabilité repose fortement sur les *services* qu'elles sont capables de fournir (les services utiles ont été décrits dans la section 5.5.2 page 163), mais également sur une gestion fine et correcte des *modifications concurrentes*. Swing, par exemple, est capable d'exposer ses widgets à deux niveaux : un niveau *modèle* spécifique au widget, et un niveau *événementiel* générique. Les modèles sont exposés avec le niveau de détail qui convient, et gèrent parfaitement la concurrence. Le niveau événementiel, sur lequel reposent les techniques de *contrôle de surface* comme le contrôle multi-pointeurs, comporte cependant quelques faiblesses.

Premièrement, chaque widget Swing gère ses propres événements positionnels et peut donc être contrôlé indépendamment, mais certains liens implicites entre ces widgets induisent des comportements inappropriés lors d'un contrôle multi-pointeurs : par exemple, cliquer sur un widget avec un pointeur fermera un menu ouvert par un autre pointeur. Ce type de mode global trahit une gestion incomplète et câblée de l'interaction concurrente.

Deuxièmement, il n'est pas surprenant de constater qu'un widget Swing est incapable de gérer deux flux d'événements positionnels (par exemple, deux drags simultanés) : la granularité du contrôle de surface se limite donc à celle du widget, ce qui est principalement gênant pour de gros widgets. Une liste, par exemple, est gérée comme un seul widget et non comme un ensemble d'éléments. Une barre de défilement constitue également un widget monolithique.

Plusieurs stratégies peuvent être envisagées pour résoudre ce problème. Nous avons déjà proposé une façon de modifier les boîtes à outils existantes en substituant à l'architecture *M[V+C]* [FOWLER 99] une architecture $MV_{ZM}C$ [DRAGICEVIC & FEKETE 99, FEKETE & DRAGICEVIC 00]. Cette dernière permet de rendre les widgets à nouveau modulaires en éliminant la gestion des entrées dans *[V+C]* et en la remplaçant par un protocole générique permettant de manipuler individuellement chacune des zones manipulables du widget (figure 5.11, image de gauche). L'objet *[V+C]* est ainsi remplacé par l'objet V_{ZM} *(Vue/Zones/Manipulateurs)*, qui au lieu de recevoir les événements d'entrée est manipulé par un ou plusieurs *contrôleurs de vue*. Le modèle est, quant à lui, directement manipulé par des *contrôleurs de modèle* (figure 5.11, image de droite).

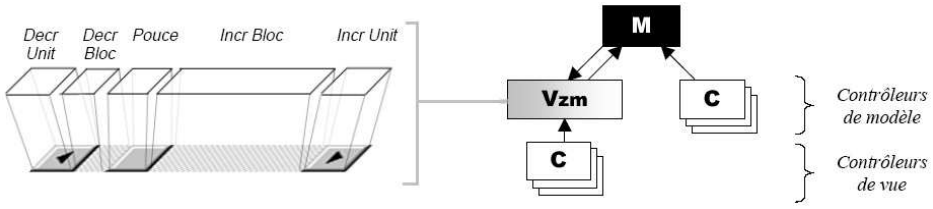


FIG. 5.11 – À gauche : les zones manipulables d'une barre de défilement. À droite : le modèle architectural $MV_{ZM}C$.

L'avantage de l'architecture $MV_{ZM}C$ est qu'elle permet de conserver la structure en classes de widgets d'une boîte à outils existante. Nous avons étendu plusieurs widgets Swing avec cette architecture et montré comment ils pouvaient être efficacement contrôlés avec des pointeurs multiples [DRAGICEVIC & FEKETE 99]. Cependant ces extensions nécessitent de modifier la machine virtuelle Java,

et en particulier de remanier les sources de Swing, qui ne sont pas un modèle de clarté ni de maintenabilité. Nous étudions actuellement une autre approche consistant à exploiter des boîtes à outils graphiques Java employant un modèle graphique structuré comme Jazz [BEDERSON *et al.* 00] ou Piccolo [BEDERSON 03] afin de contrôler les applications à l'échelle de la forme géométrique (voir également les projets liés à ICON dans la section 4.5 page 141).

5.6.3 La réutilisabilité des configurations

Il est essentiel pour le programmeur d'applications que les configurations qu'il décrit soient réutilisables. Tout d'abord, elles doivent pouvoir être chargées et exécutées sur d'autres systèmes que le sien. Ensuite, le fait de pouvoir réutiliser des configurations ou des parties de configurations d'une application à l'autre faciliterait la tâche du programmeur, mais garantirait également à l'utilisateur une certaine cohérence de l'interaction entre les diverses applications compatibles ICON.

Réutilisabilité d'un système à l'autre

Les systèmes interactifs traditionnels garantissent la compatibilité matérielle par des mécanismes d'abstraction : les pilotes de dispositifs se chargent de fournir une abstraction connue du matériel spécifique connecté à l'ordinateur. Notre approche place le problème de la réutilisabilité au second plan au profit d'outils permettant à chaque utilisateur de configurer au mieux l'interaction avec *son propre* matériel. De ce fait, afin d'exploiter au mieux les spécificités de chaque dispositif, ICON n'emploie pas d'abstractions *ad-hoc* comme les classes prédéfinies de dispositifs. Dans ICON, chaque dispositif (marque, modèle) étant distinct des autres, il n'y a pas notion de dispositifs « compatibles ».

Pour pouvoir partager les configurations, nous avons cependant intégré dans ICON un mécanisme de sérialisation et désérialisation de dispositifs extrêmement souple basé sur les *descripteurs* (section 4.3.3 page 124). Ce mécanisme permet en fonction des besoins de définir des « classes » de dispositifs interchangeables à l'aide d'un langage nommé ICSCRIPT, afin de permettre à ICON, lors du chargement de la configuration, de rechercher le dispositif qui convient le mieux parmi ceux présents. Simple mais puissant, ce langage autorise notamment les descriptions incomplètes et leur composition logique. En outre, il permet de définir des descripteurs *plus ou moins stricts*, ce qui est, comme nous l'avons vu dans la section 4.3.3 page 124, d'importance capitale car si une configuration destinée à la distribution se doit d'être tolérante sur la définition d'un dispositif d'entrée, sur un système multi-dispositifs donné il est important de retrouver les mêmes dispositifs d'une session à l'autre.

Le mécanisme de descripteurs d'ICON comporte toutefois encore certaines limites : par exemple, il permet de décrire des équivalences sémantiques entre slots de noms différents (par exemple, « le slot *stylus* sur une tablette Wintab correspond au slot *button1* sur une souris DirectInput ») mais ne permet pas d'apparier des types de données incompatibles (par exemple, « les slots *dx* et *dy* de la souris *plus une conversion en coordonnées absolues* peuvent être substitués aux slots *x* et *y* de la tablette »). Actuellement, il est nécessaire de traiter les différents cas dans des configurations séparées. Une autre faiblesse de ces mécanismes est qu'ils reposent sur une description logique des dispositifs d'entrée qui ignore la plupart des propriétés intrinsèques des dispositifs dont Buxton a suffisamment souligné l'importance (voir la section 1.2.1 page 7). Intégrer au niveau logiciel un modèle pragmatique des entrées comme celui de Buxton est un problème qui n'a à ce jour pas été résolu.

Réutilisabilité d'une application à l'autre

Dans les systèmes interactifs traditionnels, les techniques d'interaction standard (pointage, menus, widgets, etc.) sont réutilisables d'une application à l'autre et surtout, offrent une cohérence globale dans l'interface utilisateur. Dans ICON, le développeur de chaque application est libre de décrire les techniques d'interaction qu'il souhaite. L'avantage est bien sûr de pouvoir employer des interactions dédiées à la tâche, mais la réutilisabilité inter-applications reste un critère essentiel pour le programmeur et les utilisateurs.

ICON autorise à travers son modèle modulaire une certaine cohérence dans les techniques d'interaction. Plusieurs configurations d'entrée peuvent ainsi reposer sur les mêmes mécanismes de base déjà décrits par les dispositifs utilitaires. En outre, les techniques d'interaction « clé en main » peuvent être facilement réemployées d'une application à l'autre. La façon dont ces techniques sont composées et exploitées dans une application peut varier cependant, et des techniques totalement inédites peuvent facilement être décrites. ICON ne propose pas de modèle haut-niveau pour structurer l'interaction, et est en outre indifférent quant aux conventions employées (par exemple, la sémantique du bouton droit de la souris). La cohérence de l'interaction d'une application à l'autre reste donc un problème, et si les développeurs d'applications peuvent s'inspirer des conventions usuelles de la plupart des configurations reposant sur des dispositifs standard, les conventions relatives à l'interaction post-WIMP sont encore inexistantes.

Enfin, est-il possible de fournir des configurations d'entrée qui fonctionnent dans toutes les applications ? La réutilisabilité stricte des configurations d'entrée suppose une abstraction partagée par toutes les applications interactives. Les *dispositifs de boîte à outils graphique* fournissent un tel niveau de réutilisabilité, car ils permettent de contrôler l'ensemble des applications reposant sur la même boîte à outils. Bien que ce contrôle se fasse par compatibilité, des interactions non standard peuvent néanmoins être décrites, et il est possible par exemple de proposer des configurations d'accessibilité génériques[5]. En outre, des boîtes à outils graphiques évoluées comme Jazz permettent de décrire des techniques d'interaction génériques encore plus sophistiquées.

Malgré tout, le principe de configuration générique est incompatible avec celui de l'interaction dédiée à la tâche, et seuls des modèles d'application spécifiques permettront de décrire des interactions à la fois puissantes et réutilisables. Nous avons déjà évoqué la possibilité de fournir une bibliothèque de dispositifs abstraits dédiés au domaine, dispositifs pouvant être partagés par un petit nombre d'applications de même nature (par exemple, applications de dessin ou applications 3D). Cette solution semble constituer un compromis tout à fait acceptable, qui reste encore à explorer.

5.6.4 La prise en charge effective des dispositifs dans ICON

ICON s'appuie sur un petit nombre d'APIs d'entrée existantes pour la prise en charge des dispositifs non standard (voir la section 4.2.1 page 111 pour les détails). Les dispositifs d'entrée connectés au poste de travail et qui sont compatibles avec l'une de ces APIs apparaissent dans ICON et peuvent être employés dans les configurations d'entrée. Les APIs actuellement prises en charge ne couvrent cependant pas l'ensemble des dispositifs d'entrée existants, et les autres dispositifs nécessitent d'être implémentés dans ICON avant de pouvoir être utilisés. Cette opération peut se révéler fastidieuse, mais elle reste avantageuse pour le programmeur car elle permet de tirer parti du nouveau dispositif dans

[5] Voir par exemple la configuration de la figure 4.27 page 135, qui décrit le contrôle vocal d'une barre de défilement.

n'importe quelle application interactive Java.

Notons pour finir que la multiplicité des APIs d'entrée tend nettement à disparaître au profit du standard USB [USB/HID 01], et que la prise en charge de ce standard dans Java fait actuellement l'objet d'un projet d'IBM officiellement accepté comme extension candidate au langage [MAXIMILIEN *et al.* 01] ainsi que d'un projet open-source indépendant [JUSB 03]. Les deux projets proposent déjà une implémentation Linux quasi-complète. Il y a donc fort à parier qu'à terme, la prise en charge effective des dispositifs non standard dans ICON ne constituera plus un obstacle à la description de techniques d'interaction non conventionnelles. Par ailleurs, les possibilités potentielles offertes par l'accès aux dispositifs USB dans un langage multi-plateformes rendra encore plus indispensable l'existence d'un outil approprié pour les exploiter au mieux dans les applications.

5.6.5 Les performances

ICON repose sur un modèle d'exécution réactif, bien différent du modèle classique où les événements d'entrée sont accumulés dans une file pour être traités. Le modèle événementiel est de type conversationnel : l'ordinateur est maître de l'interaction et le client attend d'être servi. Dans un modèle réactif, c'est l'environnement (donc l'utilisateur) qui est maître de l'interaction, et toute action de sa part est traitée sans délai.

La machine réactive offre par conséquent des garanties en termes de temps réel. En revanche, le modèle réactif n'est valable que tant que l'*hypothèse réactive* est vérifiée, c'est-à-dire lorsque les événements sont traités plus rapidement que la vitesse où ils arrivent. La transgression de l'hypothèse réactive a des conséquences importantes sur la fiabilité du système et sur l'interaction elle-même, car des événements peuvent être perdus.

La souplesse actuelle des pilotes de dispositifs (qui gèrent notamment des mémoires tampon) fait qu'en pratique, la réactivité minimale acceptable dépend de critères pragmatiques plutôt que techniques. Ainsi, pour la plupart des techniques d'interaction, un temps de réaction de l'ordre de 1/50 de seconde permet de conserver l'information pertinente, et correspond à une réactivité et un taux de rafraîchissement considérés comme largement acceptables du point de vue de l'utilisabilité. Ce temps doit être sensiblement diminué pour certaines techniques comme la reconnaissance gestuelle, où le débit d'informations à traiter (du moins au niveau bas) est plus important.

Le temps d'exécution d'une itération (tick) est la somme du temps de *propagation* des signaux entre les dispositifs et de leur *traitement* au sein des dispositifs. Nous avons constaté que le premier était négligeable par rapport au dernier. L'algorithme de propagation employé dans ICON et décrit en annexe C est minimaliste et performant, même implémenté en Java. Pour donner un ordre de grandeur du temps de propagation, une configuration comportant 1000 dispositifs simples (se contentant de transmettre une valeur) connectés en série s'exécute en environ 0.5 ms (taux de rafraîchissement de 1800 Hz) sur une station Silicon Graphics 540 avec un processeur cadencé à 550 MHz.

Le temps de traitement évolue linéairement par rapport au nombre de dispositifs présents dans la configuration, et est au moins égal à celui du dispositif le plus lent. Ce temps doit être maintenu le plus bas possible, ce qui nécessite un style de programmation particulier pour les dispositifs (voir section 5.6.1 page 166). Dans l'hypothèse où l'instanciation d'objets est minimale et où les traitements coûteux sont lancés dans des fils d'exécution séparés, ICON s'exécute en « temps-réel ».

Les performances dépendent également des ressources processeur disponibles. Le fil d'exécution

d'ICON est prioritaire par rapport à celui qui gère le rendu graphique. Cependant, afin de permettre au rafraîchissement graphique de se faire, ICON permet de spécifier une fréquence d'exécution afin de laisser entre les ticks l'initiative aux autres processus. Le temps d'exécution de chaque tick est mesuré afin de savoir si à un moment l'hypothèse réactive est violée. Il est également possible d'analyser les performances d'une configuration d'entrée, et de déterminer la contribution de chaque dispositif au temps d'exécution.

5.7 Les utilisateurs d'ICON

Le modèle des dispositifs en cascade constitue, nous en sommes certains, une bonne infrastructure pour un outil de configuration à la fois puissant et naturel d'utilisation. Il exploite le concept de *connexion logicielle*, métaphore qui étend naturellement le paradigme de la connexion physique (section 3.2.1 page 94). Les deux entités essentielles pour une application configurable y sont exposées, à savoir les *dispositifs physiques* et les *points d'entrée* vers l'application, puis complétées par la notion d'*adaptateurs*.

L'objectif de l'éditeur interactif d'ICON est d'instrumenter ces concepts par les outils et les techniques d'interaction et de visualisation nécessaires afin d'offrir une configurabilité maximale à un nombre maximum d'utilisateurs. Nous ne nous attendons pas à ce que l'utilisateur moyen soit capable d'employer tous les fonctionnalités d'ICON. En revanche, ICON peut s'adapter à divers degrés d'expertise, et vise non seulement les programmeurs mais un plus large public composé d'utilisateurs avancés. Nous décrivons ce continuum d'utilisateurs dans cette section.

5.7.1 Un continuum d'utilisateurs

La table 5.7.1 page ci-contre énumère les différentes *tâches de configuration* qui peuvent être effectuées avec ICON, dans l'ordre croissant de difficulté. Chaque tâche nécessite des pré-requis qui sont spécifiés en deuxième colonne, et qui doivent être cumulés avec les pré-requis des tâches précédentes. Nous donnons également une idée des types d'utilisateurs que nous pensons capables d'effectuer chaque tâche.

La tâche la plus simple (tâche **1.**) consiste à choisir parmi plusieurs configurations d'entrée existantes dans une application interactive, par exemple remplacer une configuration standard par une configuration conçue pour des utilisateurs handicapés. Cette tâche peut être effectuée par n'importe quel utilisateur de l'application, pourvu que les configurations disponibles y soient correctement listées et documentées.

Les tâches suivantes, destinées à des catégories particulières d'utilisateurs, sont décrites par la suite.

5.7.2 Les utilisateurs avancés

Comme nous l'avons vu dans le chapitre 4, il est possible d'interrompre temporairement une application pour ouvrir sa configuration d'entrée dans l'éditeur interactif. Cette option est destinée aux *utilisateurs avancés*, c'est-à-dire à la catégorie des utilisateurs qui sont prêts à consacrer du temps pour personnaliser les outils qu'ils emploient (voir la discussion à ce sujet dans la section 1.4.1 page 28).

	Tâche	Pré-requis	Utilisateurs visés
1.	Sélection parmi plusieurs configurations d'entrée disponibles.	Aucun.	Tout utilisateur.
2.	Paramétrisation d'un dispositif.	Localisation du dispositif dans la configuration.	Utilisateurs avancés.
3.	Permuter des dispositifs ou des slots compatibles.	Compréhension du modèle de slots et des compatibilités de types.	
4.	Connexion directe d'un dispositif d'entrée à des dimensions compatibles de l'application.	Connaissance minimale des dispositifs d'entrée disponibles et des dispositifs de l'application.	
5.	Insertion de dispositifs de traitement supplémentaires.	Compréhension minimale du paradigme de flot de données et connaissance des principaux dispositifs de traitement disponibles.	
6.	Connexion d'un dispositif d'entrée à des dimensions incompatibles de l'application.	Maîtrise du paradigme de flots de données et bonne connaissance des dispositifs de la bibliothèque.	Scientifiques / Informaticiens.
7.	Création ou adaptation d'une configuration d'entrée complète.	Connaissance étendue des dispositifs de la bibliothèque et notions de programmation JavaScript.	
8.	Extension de la bibliothèque d'ICON par ajout de nouveaux dispositifs d'entrée ou de traitement.	Connaissance du langage Java et de l'interface `Device`.	Développeurs.

TAB. 5.1 – Tâches de configuration qui peuvent être effectuées avec ICON, de la plus facile à la plus difficile.

Selon son degré d'expertise, nous pensons qu'un utilisateur avancé peut accomplir une ou plusieurs des tâches qui suivent.

La tâche **2.** consiste à personnaliser un dispositif en ajustant ses paramètres. Un exemple consiste à changer le vocabulaire d'un dispositif de commande vocale (voir la configuration-exemple de la figure 4.27 page 135). Cela suppose, une fois la configuration d'entrée ouverte dans l'éditeur interactif, que l'utilisateur soit capable de localiser le dispositif à paramétrer dans cette configuration. Cette tâche est facilitée par la labelisation et la documentation des dispositifs, ainsi que par l'utilisation d'infobulles.

La tâche suivante (tâche **3.**) consiste à intervertir des dispositifs ou des slots compatibles. Il peut s'agir de remplacer une souris par une tablette graphique pour contrôler le curseur, remplacer une technique d'interaction prédéfinie par une autre, ou encore redéfinir des raccourcis clavier. Les changements de connexion sont aisés à effectuer grâce aux techniques de manipulation directe employées dans l'éditeur interactif.

La tâche **4.** consiste à employer un dispositif d'entrée dans une configuration existante en le connectant directement à des dimensions de l'application. Il peut s'agir d'assigner des boutons d'une souris étendue à des commandes, de contrôler le zoom d'un document avec la molette de la souris, ou de connecter la pression d'une tablette à la taille de la brosse dans un logiciel de dessin (configuration de la figure 4.24 page 134). Les connexions sont directes, et la seule difficulté ici réside dans les éventuelles incompatibilités de domaines, qui nécessitent l'emploi du dispositif de remise à l'échelle *LinearFunc*.

La tâche **5.** consiste à étendre une configuration d'entrée en ajoutant des traitements de données simples, comme par exemple insérer un filtre passe-bas pour lisser les déplacements du curseur (configuration de la figure 4.30 page 137). Ce type de tâche nécessite la connaissance préalable de quelques dispositifs de traitement prédéfinis, et la compréhension du fonctionnement général du flot de données pour être capable d'insérer le dispositif au bon endroit.

5.7.3 Utilisateurs scientifiques et informaticiens

Par rapport aux tâches vues précédemment, la tâche **6.** est plus complexe. Pour connecter des dispositifs de nature différente, des dispositifs intermédiaires de traitement et de contrôle doivent être employés, ce qui ajoute à la complexité. Par exemple, connecter une dimension continue comme la pression d'un stylet à un canal booléen comme le clic nécessite l'emploi et la paramétrisation d'un seuil. De même, spécifier des valeurs continues avec seulement deux boutons nécessite l'utilisation de techniques d'incrémentation/décrémentation.

Notons que les dispositifs utilitaires de haut niveau fournis par la bibliothèque d'ICON permettent de ramener la difficulté de la plupart des situations courantes à celle d'une tâche de type **4.** ou **5.** : les dispositifs *adaptateurs* permettent de relier entre eux des slots ou des dispositifs d'un type particulier, par des techniques génériques simples comme l'incrémentation/décrémentation précédemment évoquée, des techniques plus avancées comme le pointage au clavier (configuration de la figure 4.26 page 135), ou encore de véritables techniques d'interaction comme le dispositif *QuikWriting* (configuration de la figure 4.26 page 135).

Néanmoins, les adaptateurs prédéfinis ne répondent pas à toutes les situations, et spécifier de tels mécanismes avec des opérateurs simples peut se révéler extrêmement difficile pour des personnes n'ayant jamais travaillé sur des systèmes à flot de données. La construction d'une configuration d'entrée complète ou la modification de parties importantes d'une configuration existante l'est encore plus (tâche **7**). C'est pourquoi ces deux types de tâches sont réservés à des informaticiens sachant programmer ou des scientifiques ayant une expérience des systèmes à flots de données. Dans certains cas, l'utilisation de dispositifs programmables (voir section 4.3.1 page 118) peut remplacer avantageusement la composition graphique.

5.7.4 Développeurs

Enfin, il peut se révéler nécessaire lors de la construction d'une configuration d'entrée d'implémenter des dispositifs initialement non présents dans la bibliothèque (tâche **8.**). Ce dernier type de tâche est bien évidemment réservé aux développeurs, qui désirent enrichir la bibliothèque de nouveaux adaptateurs et techniques d'interaction ou prendre en charge de nouveaux dispositifs d'entrée.

L'ensemble des tâches que nous avons décrites jusqu'ici sont des tâches de configuration. Cette liste doit être complétée par deux autres tâches : d'abord celle qui consiste à implémenter une application compatible avec ICON (tâche **9.**), et qui nécessite une connaissance minimale de la librairie de programmation d'ICON et de ses principaux concepts (une section complète lui sera consacrée). Ensuite, celle qui consiste simplement à utiliser cette application interactive (tâche **0.**), et qui ne nécessite absolument aucune connaissance d'ICON. Pour cette dernière tâche, une documentation utilisateur précisant comment utiliser l'application toutefois peut s'avérer nécessaire, en particulier pour des configurations d'entrée qui décrivent des techniques non standard.

5.7.5 Le cycle de vie d'une configuration

Nous avons vu dans cette section comment ICON s'adresse à tout un continuum d'utilisateurs, du développeur d'applications à l'utilisateur final. Nous résumons ici le rôle respectif de chacun.

Le *développeur d'applications* produit l'application interactive et les dispositifs propres à cette application, et fournit une configuration par défaut, ainsi qu'un ensemble prédéfini de configurations d'entrée destinés à assurer une contrôlabilité et une accessibilité minimales.

L'*utilisateur final* peut utiliser l'application de manière standard, ou remplacer la configuration par défaut par celle qui lui convient. Il peut également, selon son *degré d'expertise*, personnaliser de manière plus ou moins poussée des configurations existantes ou en construire de nouvelles, pour éventuellement les partager avec d'autres utilisateurs. Les utilisateurs avancés peuvent également assister des utilisateurs dans la personnalisation de l'interaction.

Enfin, certains *développeurs* peuvent également de façon indépendante contribuer à enrichir la bibliothèque d'ICON de nouveaux dispositifs, afin de prendre en charge de nouvelles techniques d'interaction ou des dispositifs d'entrée non conventionnels.

5.8 Positionnement de notre approche et perspectives

Dans cette section, nous positionnons ICON par rapport aux principales autres approches, dans le but de dégager les points communs et les complémentarités, et d'évaluer l'originalité d'ICON par rapport aux travaux existants. Nous proposons ensuite différentes pistes de recherche pour des travaux futurs.

5.8.1 ICON et les autres approches

ICON et le modèle *MVC*

La plupart des modèles de référence (voir section 2.2 page 36) décrivent un composant *présentation* ou *interaction* qui gère à la fois les entrées et l'affichage. Le modèle *MVC*, tout comme celui d'ICON, sépare l'interaction en entrée (gérée par le *contrôleur*) de l'interaction en sortie (gérée par la *vue*). L'objectif d'ICON est de décrire la partie *contrôleur* de la triade, sans s'occuper de la structure des deux autres composants.

La figure 5.12 page suivante illustre l'architecture d'ICON dans le paradigme *MVC*. La *configuration d'entrée* constitue le *contrôleur*, hormis ses *dispositifs d'application* qui décrivent chacun une partie du *modèle* et/ou de la *vue* de l'application interactive. Les dispositifs d'application sont en quelque sorte l'*interface* entre le contrôleur et le reste de l'application. Une configuration d'entrée peut également, avec des dispositifs de *feedback*, décrire des *vues* indépendantes. Les *dispositifs de boîte à outils* ne sont pas explicitement représentés dans ce schéma ; nous les considérons ici comme des dispositifs d'application qui se distinguent uniquement par leur caractère générique.

L'objet *M+V* de la figure décrit l'application interactive de façon monolithique, sans distinguer son noyau fonctionnel de sa représentation. En pratique bien sûr, le modèle *M* et la vue *V* peuvent être explicitement séparés, de même qu'ils peuvent être structurés en objets *M+V* plus petits. Les dispositifs d'application créent déjà une certaine structuration en encapsulant des objets *M+V* individuels,

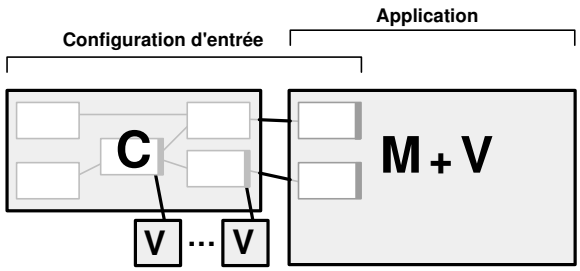


FIG. 5.12 – ICON dans le paradigme *MVC*. Les dispositifs de la partie *M+V* sont les *dispositifs d'application* de la configuration d'entrée. Les vues multiples, à gauche, matérialisent les *sorties implicites* des autres dispositifs.

qui peuvent refléter ou non une structure pré-existante dans l'application. Les *widgets* constituent un exemple de structure pré-existante exploitée par les dispositifs de boîte à outils.

Notons pour finir que les dispositifs d'application ne sont pas tous destinés à être utilisés en « bout de configuration ». Certains d'entre eux, les *points de retour*, peuvent comporter des slots de sortie et être employés à n'importe quel niveau de la configuration (notre schéma n'en comporte pas pour des raisons de lisibilité). Des dispositifs comme le *Pick* peuvent ainsi être employés pour *réinjecter* dans la configuration de l'information *en provenance de l'application*.

ICON et les modèles à événements

Les outils commerciaux de développement d'interfaces et la majorité des boîtes à outils Post-WIMP proposées par la recherche (voir 2.6 page 60) reposent sur des modèles à événements pour la gestion des entrées. ICON emploie un modèle à flot de données réactif où la notion d'événement est absente, et où le paradigme de développement est par conséquent tout autre. Les mécanismes comme l'aiguillage, l'abonnement et les callbacks, en particulier, sont remplacés par des connexions directes entre modules.

Les deux approches ne sont cependant pas incompatibles, et dans certains cas elles peuvent se révéler complémentaires[6]. Le modèle d'ICON, nous l'avons vu, est particulièrement adapté à la description de techniques d'interaction réactives et de bas-niveau. Certains modèles d'interaction, en particulier l'interaction multimodale, nécessitent des notions de plus haut-niveau, comme celles des événements ambigus et des stratégies de médiation employées par *Multimodal Subarctic* (voir section 2.6.2 page 68). Néanmoins, toute interface, y compris les interfaces multimodales, comporte une partie réactive de bas-niveau. Une interface gestuelle, par exemple, a besoin d'accéder aux dispositifs d'entrée physiques, de produire en temps réel des traces et de les afficher.

Nous pensons que tout paradigme d'interaction gagnerait à reposer sur une infrastructure de type ICON. Une telle infrastructure procure de nombreux avantages, et garantit en particulier une grande souplesse par rapport aux entrées. Pour reprendre l'exemple de l'interaction gestuelle, d'autres dispositifs que la souris ou la tablette (par exemple, un dispositif de suivi oculaire) sont susceptibles de produire une trace. Même des dispositifs de nature non positionnelle peuvent être exploités avec des algorithmes et des mécanismes de reconnaissance gestuelle : une succession de touches au clavier est

[6]Par exemple, ICON permet facilement de décrire des dispositifs *« générateurs d'événements »* qui, à partir d'attributs simples transmis en entrée, et à chaque signal sur un slot `send`, instancient des événements pour les placer dans une file.

interprétable comme un geste, par exemple. Le modèle d'ICON, et sa notion d'adaptateurs en particulier, facilite et même encourage les utilisations multiples des dispositifs d'entrée. Au contraire, l'approche événementielle, qu'elle soit ou non ouverte aux entrées non conventionnelles, requiert une classification des dispositifs et l'attribution d'un rôle figé à chacun d'entre eux.

Dès lors, il ressort une question essentielle, à savoir *jusqu'à quel niveau l'interaction doit être décrite avec ICON ?* Nous n'avons pas étudié les manières de combiner notre modèle avec des architectures Post-WIMP de plus haut-niveau. Cependant, nous avons montré comment ICON pouvait permettre de décrire des techniques d'interaction de type « instrumental » dans un certain niveau de granularité. Cette expérience nous a notamment permis de réaliser qu'il serait par trop simplificateur de décomposer la gestion de l'interaction en couches figées. En effet, certains éléments contrôlables d'une application interactive (ou *points d'entrée*, dans notre terminologie) sont de bas-niveau et peuvent être manipulés avec des techniques de contrôle direct. Par exemple, la plupart des widgets sont des dispositifs « virtuels » qui peuvent, à la manière des *Phidgets*, être directement associés à des dispositifs réels. C'est également vrai pour certaines structures propres à l'application. Des parties de l'interaction peuvent par conséquent être intégralement décrites avec des flots de données réactifs, des dispositifs physiques jusqu'à l'application. Une architecture interactive employant un modèle du type ICON comme couche basse devrait par conséquent autoriser ce type de « court-circuitage », sous peine d'ignorer les techniques de contrôle direct ou de les décrire avec des mauvaises abstractions.

ICON et les systèmes de transition

Les réseaux de Pétri et les automates à états sont des approches *orientées contrôle*, orthogonales à la notre. Elles décrivent particulièrement bien les mécanismes de multiplexage temporel et les modes employés dans les interfaces conventionnelles. À l'inverse, les interfaces Post-WIMP de type instrumental comportent relativement peu de contrôle et sont plutôt caractérisées par des *flots de données*. Cependant, à moins de disposer d'un dispositif physique pour chaque objet d'intérêt, toute interface comporte nécessairement une partie contrôle. En outre, pour des raisons d'accessibilité, nous nous devons de prendre en charge les situations d'entrées appauvries. Ce que nous faisons actuellement, mais en encapsulant les techniques d'accessibilité dans des adaptateurs.

S'il est difficile d'intégrer le contrôle et les données dans le même paradigme, deux types d'approches hybrides sont envisageables : au niveau macroscopique à l'*extérieur* d'ICON et au niveau microscopique à l'*intérieur* des dispositifs. La première approche consisterait à rendre les configurations d'entrée dynamiques et contrôler les reconnexions dans des automates ou des réseaux de Pétri, à la manière de VRED (voir section 2.7.3 page 88). Bien que nous ayons déjà motivé une approche essentiellement statique et que nous estimons que le problème de l'hybridation n'a pas encore été résolu de façon convaincante au niveau du langage visuel, ce type d'approche mérite d'être étudié.

L'approche consistant à décrire une partie des comportements des dispositifs avec des machines à états nous paraît une piste prioritaire. Un langage impératif comme Java permet d'implémenter facilement des dispositifs, mais ne dispose pas des outils adaptés pour décrire des machines à états complexes. Une extension syntaxique comme celle proposée par [BLANCH 02] permettrait d'employer les approches à flot de données et à flot de contrôle de manière orthogonale et réellement complémentaire. L'agencement visuel de blocs reposerait ainsi sur un paradigme à flot de données, et les mécanismes internes à ces blocs seraient décrits avec un langage programmation conventionnel mais sachant prendre en charge les transitions d'états.

Les approches analogues

ICON se situe à la frontière de plusieurs approches. Tout d'abord, le principe de *flots de données* sur lequel il repose a été exploité dans de nombreuses applications, principalement dans les domaines de traitement et d'analyse d'images et de composition musicale. Ensuite, ICON emploie un *langage visuel* pour décrire l'interaction, comme le font beaucoup de logiciels commerciaux et d'outils produits par le monde de la recherche. Cependant, les nombreuses approches du type *Visual* C^{++} de *Microsoft*, et y compris celle de *Garnet/Amulet* (voir section 2.6.1 page 63) se concentrent principalement sinon exclusivement sur l'agencement de composants visuels. Les éditeurs de comportement comme *Thinglab* ou *Fabrik* (voir section 2.7.2 page 80) peuvent permettre de décrire certains aspects touchant à l'interaction en entrée, mais sont plus adaptés à la partie modèle (synchronisation entre plusieurs modèles et entre modèles et vues) et graphique (maintien de contraintes géométriques) de l'application.

L'approche consistant à utiliser un langage visuel basé sur un modèle à flot de données pour décrire l'interaction en entrée a été très peu explorée dans le domaine de l'interaction 2D. L'un des rares exemples est la boîte à outils *Whizz* et l'éditeur *Whizz'Ed* (section 2.7.2 page 83). Si ce langage visuel a été principalement créé pour décrire des interfaces animées et des interactions simples, son extension à l'interaction bimanuelle (section 2.6.2 page 70) donne des exemples de configuration d'entrée assez proches de ce que l'on pourrait construire avec ICON. *Whizz'ed* donnait déjà une bonne idée de la puissance du modèle à flot de données pour décrire le contrôle d'applications hautement interactives. ICON approfondit et étend cette approche, tout d'abord en posant les principes d'un modèle d'entrée basé sur des dispositifs généralisés regroupant dispositifs physiques, points d'entrée et adaptateurs, ensuite en proposant un modèle d'exécution basé sur les langages réactifs. Enfin, ICON explore plus loin ce paradigme, en exploitant des dispositifs d'entrée multiples et bien d'autres techniques d'interaction non-standard.

Si notre approche est novatrice dans le monde de l'interaction 2D, elle peut néanmoins sembler familière aux développeurs d'applications 3D. Nous avons vu que la plupart des boîtes à outils 3D reposaient sur un modèle d'entrée à flot de données, et que quelques unes proposaient des éditeurs visuels comparables au notre. C'est notamment le cas de *Virtools Dev* (voir section 2.7.3 page 86). Ces outils sont bien adaptés à la description d'interactions 3D, et plus particulièrement à la manipulation directe d'objets 3D avec des dispositifs sophistiqués. ICON applique à l'interaction 2D les techniques qui font le succès des applications 3D, à savoir l'usage de dimensions physiques multiples dans des techniques contrôle direct. En outre, ICON prend en charge des dispositifs plus divers et intègre des techniques 2D que les outils 3D étaient incapables de décrire, notamment parce que ces techniques sont beaucoup plus variées et complexes, comportent des modes et des feedbacks, et opèrent sur des objets de nature bien plus diverse que les attributs d'objets 3D.

5.8.2 Quelques perspectives

ICON, en tant que travail exploratoire, ouvre de nombreuses voies. Nous en avons souligné quelques-unes sur la figure 5.13 page ci-contre. Nos contributions y sont évoquées en gris, à savoir le paradigme de cascades de dispositifs (en bas) et ses applications successives : le modèle ICOM, la boîte à outils et l'éditeur d'ICON, les configurations d'entrée que nous avons construites et enfin leurs applications. Ces contributions sont agencées selon l'axe *application*, qui constitue une première direction d'exploration possible pour les perspectives. Les deux autres axes sont la *validation* de notre approche (cet

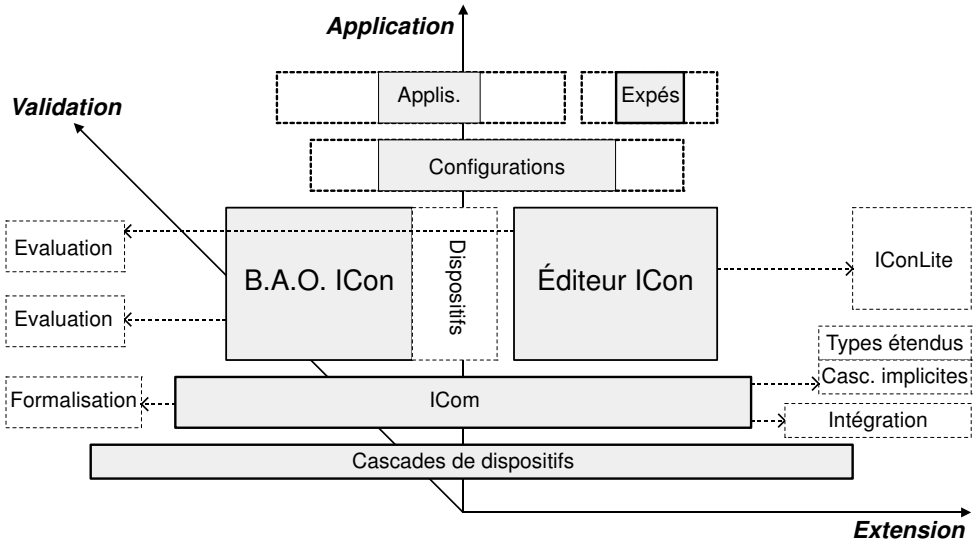


FIG. 5.13 – Perspectives.

axe dépend en partie des applications développées, d'où son orientation diagonale) et l'*extension* de celle-ci.

Le modèle ICOM décrit les mécanismes de l'infrastructure data-flow d'ICON. Plusieurs orientations sont possibles pour celui-ci : d'abord, une **formalisation** permettrait d'une part de valider la fiabilité de notre modèle, d'autre part de vérifier des propriétés sur les configurations. Nous avons déjà évoqué la possibilité d'exploiter des outils de vérification de langages réactifs comme Esterel [FEKETE *et al.* 98, FEKETE & DRAGICEVIC 00]. Diverses voies sont possibles quant à l'extension du modèle ICOM : l'**intégration** avec des modèles existants, en particulier des formalismes orientés-contrôle et des modèles d'interaction plus haut-niveau (voir 5.8.1 page 179) ; la prise en charge de **types étendus** pour faciliter la construction de configurations d'entrée et de **cascades implicites éditables** pour exposer efficacement les modèles physiques des dispositifs. Ces deux dernières idées seront développées plus loin.

En dehors de son infrastructure ICOM, ICON peut être divisée en deux parties : la partie boîte à outils qui décrit la librairie extensible de dispositifs, et l'éditeur interactif de configurations. Chaque de ces deux parties peut être validée par une **évaluation** : la première pour prouver la pertinence de l'outil vis-à-vis du programmeur, et la seconde pour déterminer l'utilisabilité du configurateur interactif. Nous avons déjà constaté que les deux développeurs que nous avons suivis étaient capables, après un temps d'assimilation, de produire rapidement des dispositifs d'application à chaque fois que cela s'avérait nécessaire, et de maîtriser toutes les potentialités de l'éditeur interactif. Nous ne nous attendons cependant pas à ce qu'un utilisateur moyen emploie l'éditeur de la même manière. Au lieu de cela, nous avons montré en quoi l'utilisateur, selon ses compétences, pouvait tirer parti de ce langage visuel pour des tâches simples. Une évaluation permettrait de déterminer plus précisément en quoi consistent ces tâches. Nous nous doutons cependant que la mise en place de telles expérimentations serait loin d'être triviale, en particulier s'il s'agit d'évaluer la pertinence d'ICON par rapport à d'autres outils. Ainsi, si les boîtes à outils conventionnelles permettent de développer facilement des interfaces WIMP, elles ne prennent pas en charge la majeure partie des techniques d'interaction qu'ICON permet

de décrire. La comparaison est difficile, faute de cibles de comparaison.

La bibliothèque d'ICON, bien que déjà très complète, gagnerait cependant à être étendue avec de nouveaux **dispositifs**. Les principaux dispositifs système que nous projetons d'implémenter, dans le but de couvrir une plus vaste palette d'entrées non standard, sont les périphériques MIDI, USB, et l'analyse vidéo. Nous avons depuis peu à notre disposition un certain nombre de boîtes à potentiomètres MIDI dont l'exploitation dans ICON nous semble très prometteuse. L'interface HID des périphériques utilisateur USB, en plus d'être générale, est très détaillée et intègre bon nombre d'informations pragmatiques sur le dispositif physique (même si malheureusement, les constructeurs n'implémentent pas toujours cette interface entièrement). L'interaction multidigitale [REKIMOTO 02], maintenant réalisable avec des dispositifs tactiles matriciels du commerce [TACTEX 03, FINGERWORKS 03], nous paraît également avoir un fort potentiel. Nous projetons également d'ajouter d'autres dispositifs utilitaires à ICON, dont divers adaptateurs d'accessibilité (techniques à simple bouton, notamment), d'autres techniques d'interaction prêtes à l'emploi, et des dispositifs dédiés au feedback reposant sur un modèle graphique multicouche similaire à [FEKETE 96B]. Enfin, nous avons déjà évoqué un projet en cours dont le but est de décrire de nouveaux dispositifs de boîte à outils pour contrôler les applications zoomables développées avec Piccolo [BEDERSON 03]. L'objectif de ces dispositifs est l'intégration complète d'ICON à une boîte à outils graphique avancée.

Nous avons déjà proposé dans ce mémoire quelques améliorations pour le configurateur interactif d'ICON : par exemple, l'annotation des configurations à des fins de documentation et une technique de zoom sémantique pour naviguer dans les dispositifs composites. Les utilisateurs non-experts pourraient également bénéficier d'une version allégée du configurateur interactif qui leur serait spécialement destinée : nous développerons les idées à la base d'ICONLITE par la suite.

À un niveau au-dessus se situent les **configurations** d'entrée construites avec ICON. Bien qu'elles soient déjà nombreuses, bien d'autres techniques peuvent être décrites avec les dispositifs existants. En particulier, nous pensons que le potentiel des techniques d'interaction de bas-niveau type « pointage augmenté » mérite d'être davantage exploré. En outre, au fur et à mesure qu'ICON se verra enrichir par de nouveaux dispositifs, d'autres configurations exploitant ces dispositifs viendront s'ajouter à celles existantes.

L'objectif des configurations d'entrée est de décrire l'interaction avec des **applications**. À ce jour, toutes les applications reposant sur Swing peuvent être contrôlées avec ICON (nous avons également décrit le contrôle partiel de Microsoft Word et de Jazz, mais uniquement à titre expérimental). En dehors des applications-jouets comme ICONDraw, deux projets d'applications employant ICON ont été évoqués (voir section 4.5 page 141). Nous projetons de décrire le contrôle d'applications de natures plus variées et en particulier des applications gourmandes en entrées tels que les jeux vidéo et les applications d'animation 3D. Les visages animés comme ceux des agents Haptek [HAPTEK 03] notamment, comportent un grand nombre de dimensions qui peuvent être contrôlées simultanément. Nous espérons également que notre distribution d'ICON sera employée dans des projets de développement extérieurs. Enfin, comme nous l'avons vu dans la section 4.5 page 141, ICON a également servi à monter une **expérimentation** visant à comparer plusieurs techniques d'interaction. Nous pensons qu'ICON pourra être d'une aide non négligeable dans ce type de tâche, car il offre un accès aisé aux dispositifs non-standard et permet de prototyper rapidement des techniques d'interaction, de les affiner et d'en étudier plusieurs variantes. En outre, ICON rend explicite toute la chaîne de traitements, et la représentation visuelle qu'il donne de cette chaîne est un outil de compréhension qui peut être mis à profit pour illustrer des études comparatives.

Nous terminons cette section par trois perspectives d'extension que nous venons d'évoquer et qui méritent d'être un peu plus développées : les types étendus, les cascades implicites éditables, et ICONLITE.

Types étendus

Il existe de nombreuses raisons d'étendre notre système de typage avec des types plus restrictifs. Tout d'abord, les remises à l'échelle sont fréquentes dans ICON, et il est difficile d'employer l'opérateur de transformation linéaire sans connaître les domaines de départ et d'arrivée[7]. Il nous semblerait par conséquent utile d'inclure les domaines dans les types de slot, information qui pourrait par exemple être exploitée par des adaptateurs de domaine automatiques, ou comme pré-conditions dans des dispositifs d'application.

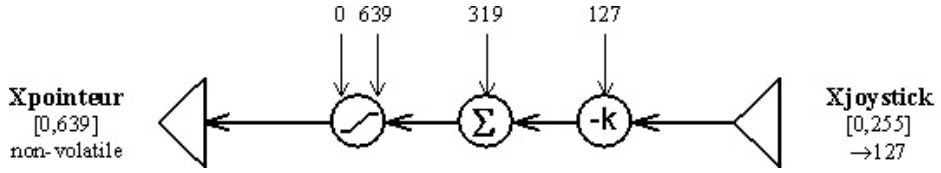


FIG. 5.14 – Utilisation de la valeur au repos pour justifier une technique de contrôle du second ordre. La dimension *x* de la manette (à droite) contrôle la dimension *x* d'un pointeur (à gauche) après une opération de recentrage, une intégration, et un seuil. Une connexion directe serait permise avec un typage simple, mais enfreindrait ici les pré-conditions du pointeur (domaine et non-volatilité) [DRAGICEVIC 98].

Les pré-conditions et post-conditions pourraient être généralisées à bien d'autres propriétés, notamment dans le but de prendre en compte certaines caractéristiques pragmatiques des dispositifs [BUXTON 83]. Ces propriétés doivent pouvoir être aisément propagées à travers les dispositifs. C'est le cas des *valeurs particulières*, qui peuvent dans certains cas être transformées par la fonction d'exécution[8]. Dans [DRAGICEVIC 98], nous décrivons des propriétés pragmatiques qui rentrent dans ce cadre. Un slot, par exemple, peut conserver sa valeur lorsque l'utilisateur n'exerce plus d'action sur les dispositifs physiques, ou bien revenir à une *valeur au repos*. Cette valeur au repos permet de spécifier des pré-conditions extrêmement intéressantes dans les dispositifs d'application (figure 5.14). D'autres propriétés comme la *précision* et la *latence* peuvent être également prises en compte, et propagées de façon spécifique.

Les types étendus avec des pré-conditions et des post-conditions permettraient à la fois de proposer des adaptateurs évolués facilitant significativement la construction de configurations, et de vérifier certaines propriétés dans la configuration d'entrée. Par exemple, une précision minimale pourrait être requise en entrée d'un pointeur ou d'un dispositif gestuel. Malgré les nombreuses applications, cette approche demande un vaste travail et pose plusieurs problèmes : tout d'abord l'utilisation de types étendus complique l'implémentation des dispositifs qui doivent calculer explicitement certaines

[7] Nous avons en partie résolu ce problème en standardisant les domaines lorsque c'était possible : les coordonnées écran sont employées pour les valeurs positionnelles (la tablette produit des position écran en valeur flottante), et la plupart des autres valeurs bornées, comme la pression du stylet, sont normalisées dans le domaine $[0,1]$.

[8] Les valeurs particulières peuvent être propagées automatiquement à travers des dispositifs déterministes du premier ordre (sans mémoire). Les bornes *min* et *max* du domaine sont des valeurs particulières, mais qui nécessitent un traitement spécifique lorsque le dispositif décrit une fonction non strictement monotone.

post-conditions[9]. En outre, les post-conditions ne peuvent pas toujours être déduites précisément, et deviennent rapidement vagues au fur et à mesure des traitements. Enfin, les informations concernant les dispositifs d'entrée physiques (par exemple, la valeur au repos) ne sont pas toujours disponibles.

Cascades implicites éditables

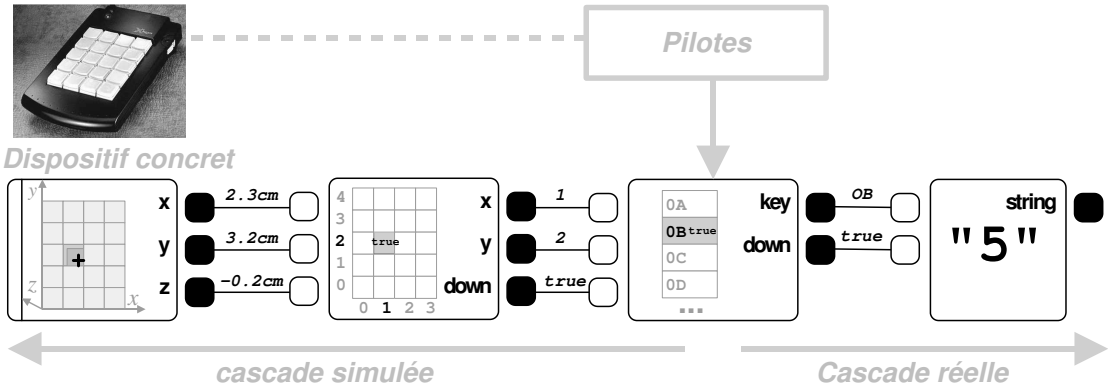


FIG. 5.15 – Exemple de cascade implicite pour une boîte à boutons : le pilote de dispositif produit des événements *down/up* avec des codes touche (troisième dispositif en partant de la gauche), et chaque événement *down* est converti en caractère. Il est possible d'imaginer une série de traitements implicites menant à la production des événements *down/up* : par exemple, lorsque l'utilisateur presse un bouton, la nouvelle position (x,y,z) de ce bouton (qui est aussi celle du doigt) est émise, puis discrétisée, et enfin convertie en code touche.

Lorsqu'un dispositif d'entrée ICON fournit une vue logique assez éloignée du modèle concret du dispositif, il serait intéressant à des fins d'explication de rendre le dispositif concret visible dans la configuration, et de reproduire la cascade implicite (ou une cascade possible) de traitements en aval de ce dispositif (figure 5.15). Le dispositif de reconnaissance vocale, par exemple, pourrait apparaître comme un dispositif de traitement connecté à un microphone physique.

La cascade implicite peut même être rendue *modifiable* si les dispositifs qui la composent sont réels et non plus simulés, et qu'ils sont *inversibles*. Un dispositif de fonction de transfert f est *inversible* s'il existe un dispositif de fonction de transfert f^{-1} telle que $f \cdot f^{-1} = Id$ (voir également [ACCOT *et al.* 97]). Or supposons que $f = g \cdot h$ désigne une cascade implicite divisée en une partie non-inversible g et une partie inversible h. Si la partie h est modifiée en h_2, il suffirait pour prendre en compte cette modification d'appliquer la fonction de transfert $h^{-1} \cdot h_2$ en sortie du dispositif logique. Ainsi, dans l'éditeur graphique, les dispositifs de g seraient non modifiables mais la cascade h pourrait être modifiée comme tout élément de la configuration. Le traitement réel $h^{-1} \cdot h_2$ resterait cependant invisible à l'utilisateur.

Les applications potentielles des cascades implicites éditables sont nombreuses. Sur la configuration de la figure 5.15 par exemple, il serait possible de se servir de la vue matricielle de la boîte à boutons pour calculer la moyenne des *(x,y)* enfoncés et contrôler grossièrement un pointeur. Le dispositif concret pourrait fournir des informations spatiales également utiles. Un modèle de manette intégrant les différentes grandeurs physiques (déplacement réel, force exercée sur le manche), par

[9] Le calcul des post-conditions pourrait cependant être pris en charge par un ensemble restreint de dispositifs atomiques prédéfinis.

exemple, permettrait de donner un sens aux valeurs numériques produites par le pilote de dispositif [DRAGICEVIC 98]. La cascade implicite pourrait également remonter jusqu'à un modèle d'utilisateur comportant des post-conditions qui décrivent ses capacités motrices (par exemple, la force maximale qu'il est capable d'exercer).

Le calcul de l'inverse devrait être à la charge de chaque dispositif, qui implémenterait une fonction du type `Device createInverse()`. L'inconvénient, là encore, réside dans la difficulté pour le développeur de programmer de tels dispositifs, en particulier si les inversions sont employées en combinaison avec des types étendus[10].

ICONLITE

Notre expérience avec ICON nous a conduit à penser qu'il est possible, en poussant plus loin l'exploitation de la métaphore de la connexion logicielle (voir section 3.2.1 page 94), de construire un éditeur de configurations à l'usage de l'utilisateur moyen, qui soit à la fois simple d'utilisation et puissant. Ceci à condition de concevoir les techniques de visualisation et d'interaction appropriées.

L'éditeur *Whizz'Ed* (voir section 2.7.2 page 83) emploie des représentations iconiques des dispositifs, ce qui facilite leur identification. Les *Phidgets* (voir section 1.3.5 page 23) exploitent une technique de configuration extrêmement naturelle, consistant à brancher un dispositif USB puis à l'associer à un widget présent à l'écran par un simple clic. Si la contrôlabilité et la configurabilité offerte par les *Phidgets* est bien en deçà d'ICON, le paradigme de connexion dynamique peut être étendu et exploité dans une version allégée d'ICON.

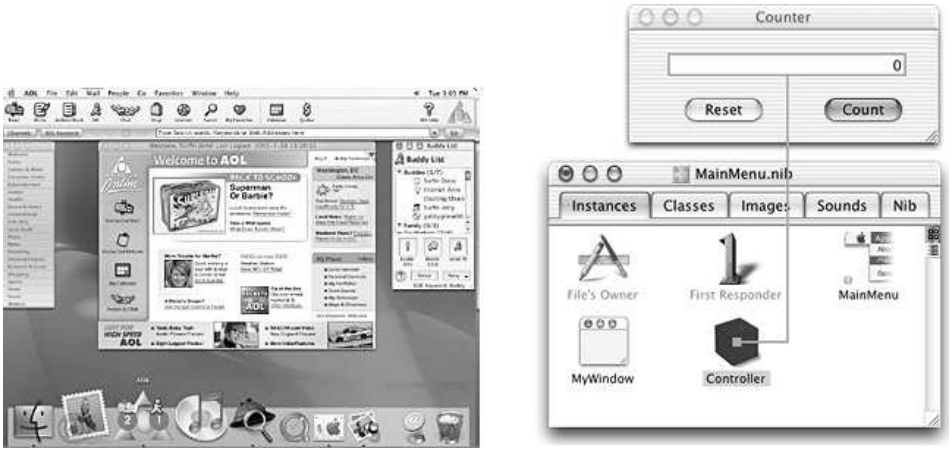


FIG. 5.16 – À gauche : le *dock* de Mac OS X (en bas de la fenêtre). À droite : *Interface Builder* d'Apple, qui permet de connecter graphiquement des widgets à leur modèle, et de visualiser ces connexions à tout moment.

Les dispositifs d'entrée connectés peuvent être représentés graphiquement dans une *barre de dispositifs* inspirée du *dock* de Mac OS X [APPLE 02] (figure 5.16, image de gauche). Lorsqu'un nouveau dispositif est branché, cette barre apparaît pour permettre d'établir une connexion avec les éléments présents à l'écran. Les connexions sont affichées en incrustation à la manière d'*Interface*

[10] Dans le cas de la boîte à boutons, par exemple, une cascade inversée produirait à partir de chaque symbole un domaine bidimensionnel.

Builder [APPLE 03] (figure 5.16 page précédente, image de droite) dans le mode de construction, et disparaissent dans le mode d'interaction. Le mode de construction peut être réactivé à tout moment pour modifier les connexions.

Une palette d'adaptateurs permettrait d'insérer des traitements supplémentaires dans les connexions (par exemple, une répétition automatique pour un bouton). Ceux-ci doivent être simples d'utilisation et contrairement à la version actuelle de notre éditeur, avoir une représentation graphique iconique explicite. Des mécanismes de connexion « intelligents » doivent pouvoir permettre l'insertion automatique d'adaptateurs (adaptateurs de domaine, notamment), ainsi que les connexions groupées de slots entre objets compatibles.

En outre, des objets d'application non visuels (par exemple, le pan ou le zoom) doivent pouvoir être réifiés et affichés, par exemple dans une palette propre à l'application. Il en est de même pour des objets système comme les pointeurs.

Deux objets compatibles (par exemple, une souris et un curseur) peuvent être directement reliés, ce qui ne nécessite pas la présence explicite de slots. En revanche, il est indispensable que les objets puissent être affichés avec plusieurs niveaux de détail. Par exemple, un clavier peut être relié à un éditeur de texte, mais des connexions de touches individuelles doivent également être possibles.

Pour finir, il serait intéressant d'explorer les différents moyens d'étendre matériellement nos systèmes afin de déplacer une partie de la tâche de configuration vers le niveau physique. Par exemple, l'organisation de la barre de dispositifs peut reproduire l'agencement spatial des prises USB, ces dernières pouvant notamment être placées sous l'écran. Chaque prise USB pourrait se voir assigner un rôle différent, et des adaptateurs matériels pourraient également être employés (des adaptateurs USB portant un identifiant reconnu par ICONLITE suffiraient). Certains utilisateurs disposeraient par exemple d'adaptateurs d'accessibilité personnels. Enfin, étant donné qu'un dispositif de pointage est toujours nécessaire pour connecter des dispositifs à des objets graphiques, un écran tactile pourrait être employé et dédié à cet effet.

Conclusion et perspectives

Rappel de la problématique

Nous avons introduit dans cette thèse le terme d'*adaptabilité en entrée* comme une combinaison de trois propriétés : la *contrôlabilité*, qui caractérise la capacité d'un système interactif à exploiter efficacement des entrées enrichies, l'*accessibilité*, qui définit sa capacité à exploiter des entrées appauvries, et la *configurabilité*, qui décrit sa capacité à pouvoir être finement personnalisée du point de vue des entrées. Ces trois propriétés sont fortement liées et constituent trois facettes d'une propriété plus générale, qui caractérise des systèmes interactifs entièrement « ouverts en entrée ». Après avoir montré en quoi il était essentiel de tendre vers ce type de système, nous avons mis en évidence le retard considérable accumulé par les systèmes interactifs grand public actuels, retard largement dû à des outils de développement figés, exclusivement basés sur le modèle de l'interaction standard (souris, clavier et techniques d'interaction WIMP). Les rares applications faisant usage d'entrées non conventionnelles, comme les jeux vidéo ou certaines applications semi-professionnelles, nécessitent une programmation artisanale extrêmement coûteuse en temps et en énergie.

Si la complexité et l'importance des problèmes liés à l'interaction en entrée sont aujourd'hui universellement reconnues par la communauté de l'interaction homme-machine, ce domaine a encore fait l'objet de peu de travaux. Nous avons présenté dans notre état de l'art les principaux modèles et outils pertinents du point de vue de l'interaction en entrée, et avons montré en quoi certaines approches constituaient une avancée dans le domaine de l'interaction en entrée, bien qu'aucun modèle ou outil à ce jour ne réponde aux exigences de l'adaptabilité en entrée sous tous ses aspects.

La spécificité de notre démarche

Les contributions de la recherche les plus pertinentes du point de vue de notre problématique sont les boîtes à outils graphiques avancées, qui se sont données pour objectif de corriger les principales faiblesses des outils de développement existants. Les contributions comme Subarctic et Garnet/Amulet montrent qu'un modèle d'interaction en entrée standard assez général peut être en partie étendu pour décrire des techniques plus évoluées. Les boîtes à outils Post-WIMP ont, de leur côté, introduit de nombreux modèles nouveaux d'interaction, spécifiquement adaptés à des paradigmes comme l'interaction gestuelle, l'usage de modalités ambiguës, l'interaction multi-utilisateurs ou encore l'interaction bimanuelle avec des outils semi-transparents. Malgré leurs apports indéniables, les boîtes à outils avancées s'appuient sur des modèles événementiels conventionnels, où seuls les dispositifs standard ou un nombre fixe de dispositifs étendus peuvent être pris en compte. Par ailleurs, ces outils exploitent ces dispositifs de manière efficace mais figée, ou au mieux difficile à étendre.

Depuis l'introduction des standards comme GKS, bien des modèles et des outils continuent d'employer des abstractions dans lesquelles les dispositifs d'entrée sont regroupés en classes d'équivalence stéréotypées. Les tâches d'interaction ont subi des simplifications analogues, notamment par l'introduction des widgets génériques. Essentiellement motivée par des problèmes de portabilité et de réutilisabilité, ce type d'approche empêche cependant de tirer parti des capacités intrinsèques de chaque dispositif et des caractéristiques de la tâche. Notre démarche se distingue des approches existantes dans le sens où elle néglige (dans un premier temps) les problèmes de portabilité pour tenter de répondre à la question suivante : sur un système concret, qui comprend un ensemble donné de dispositifs physiques, l'utilisateur peut-il efficacement tirer parti de ces dispositifs pour contrôler une application donnée ?

Il nous a paru nécessaire pour répondre à cette question de substituer aux dispositifs logiques des *dispositifs concrets*, qui apparaissent au plus bas niveau possible, c'est-à-dire comme des ensembles de canaux bruts ; de même, les objets du domaine doivent pouvoir être exposés *sans a priori sur la façon dont ils seront contrôlés*, et nous devons enfin disposer d'un moyen flexible de relier les canaux d'entrée à l'application interactive pour pouvoir la contrôler. Notre démarche s'inspire d'approches couramment employées dans les outils de développement d'applications 3D. Comme certains de ces outils, nous proposons un paradigme de programmation visuelle pour l'assignation des canaux : cette idée est pertinente dans la mesure où les canaux ne comportent pas de sémantique, et l'accès à tout nouveau dispositif physique dans un langage textuel nécessiterait des requêtes préalables à l'API de bas niveau pour connaître l'interface du dispositif. À l'inverse, un nouveau dispositif peut apparaître explicitement dans un langage visuel, auquel cas ses canaux ne nécessitent qu'une interprétation sémantique de la part de l'utilisateur pour pouvoir être exploités.

ICON et son modèle d'interaction en entrée

Le modèle d'interaction en entrée que nous avons proposé dans cette thèse est basé sur le principe de *configurations d'entrée*, dont les *dispositifs généralisés* constituent les briques de base. Les dispositifs généralisés sont des modules (agents) qui comportent des canaux en entrée et en sortie et peuvent traiter de l'information. Ils regroupent les *dispositifs système*, qui décrivent principalement des dispositifs d'entrée physiques, les *dispositifs utilitaires*, modules de traitement de données et de feedback, et les *dispositifs d'application*, qui exposent indépendamment des entrées les dimensions directement contrôlables de l'application. Une configuration d'entrée décrit la façon dont des dispositifs système sont connectés aux dispositifs d'application à travers des dispositifs utilitaires. Dans ce paradigme, les dispositifs utilitaires peuvent être vus aussi bien comme des adaptateurs que comme des « morceaux » de techniques d'interaction. Les configurations d'entrée s'appuient sur un modèle à flot de données qui permet de décrire des canaux structurés et des dispositifs composables et spécialisables. Elles emploient en outre un modèle d'exécution qui s'inspire des langages réactifs.

Le système ICON (Input Configurator) a permis à la fois de valider et d'affiner notre modèle, et de montrer la faisabilité d'un outil de prototypage et de développement pour des interfaces adaptables en entrée. ICON est une *boîte à outils d'entrées*, qui ne traite que les aspects « entrées » indépendamment des aspects « sorties », contrairement aux boîtes à outils existantes où ces deux notions sont habituellement imbriquées. Développé dans le langage Java, ICON comprend un ensemble riche et aisément extensible de dispositifs système et utilitaires, les dispositifs d'application étant déclarés par chaque application à travers un mécanisme simple. Les applications Java/Swing existantes peuvent également être contrôlées de manière générique à travers des dispositifs « de boîte à outils graphique ». ICON

peut accéder aux entrées et contrôler les applications à plusieurs niveaux, et peut être employé pour redéfinir tout ou partie de l'interaction en entrée. Il est par exemple possible d'ajouter une prise en charge des dispositifs spécialisés à une application 3D existante tout en conservant sa gestion standard des entrées.

ICON offre en outre des outils pour assurer une portabilité minimale des configurations d'entrée d'un système à l'autre, par des mécanismes qui permettent de décrire des équivalences entre dispositifs concrets sans notion de classe a priori. Les mécanismes des *descripteurs de dispositifs* sont uniques dans la mesure où ils permettent de jouer sur la finesse des descriptions pour rendre des configurations standard plus tolérantes aux différences de plate-formes avant distribution, ou au contraire leur permettre de dissocier sur une plate-forme donnée deux dispositifs très proches mais employés différemment (souris main gauche et souris main droite, par exemple).

Décrire l'interaction avec des flots de données

Notre approche à flot de données est innovante dans le domaine des applications interactives 2D, où les rares éditeurs visuels de comportement emploient comme Thinglab des systèmes à base de contraintes, principalement pour décrire des relations géométriques entre les widgets dans les applications WIMP. Seul l'outil 2D Whizz'ed a servi à décrire quelques techniques d'interaction bimanuelle par des flots de données, mais l'approche n'avait pas encore été généralisée à d'autres dispositifs et d'autres techniques Post-WIMP ou d'accessibilité. Les flots de données ont, nous l'avons vu, connu plus de succès dans le domaine de la 3D, mais les interactions que ces outils permettent actuellement de décrire sont encore limitées. ICON permet de contrôler des objets plus variés que des attributs de formes 3D, d'exploiter des dispositifs d'entrée de nature bien plus hétérogène, et permet d'employer ou construire des techniques d'interaction plus évolués que le simple contrôle direct.

Notre approche à flot de données se situe par ailleurs à l'opposé des démarches de modélisation orientées contrôle comme les réseaux de Pétri ou les automates à états, adaptés à la description du multiplexage temporel et des modes, mais non à la représentation et au traitement de l'information. La description du contrôle dans ICON est possible mais se fait généralement au détriment de la lisibilité. Son éditeur interactif fournit malgré tout des outils (raccourcis de dispositifs, zoom, encapsulation par composition) qui se sont révélés efficaces pour gérer ce type de complexité.

Nous pensons cependant que le tout-visuel a ses limites, et prônons une approche mêlant l'emploi d'opérateurs simples et de techniques d'interaction monolithiques développées dans un langage conventionnel. Notre expérience a en effet montré qu'à un certain niveau de granularité, l'interaction en entrée pouvait en grande partie être décrite comme des flux unidirectionnels de données et qu'à ce niveau, la configurabilité l'emportait avantageusement sur la complexité. Idéalement, l'emploi d'ICON devrait se limiter à l'insertion et à la paramétrisation d'adaptateurs prédéfinis comme le dispositif gestuel (qui traduit des séries de positions en commandes booléennes et produit du feedback) ou le dispositif « contrôle clavier » (qui convertit quatre canaux booléens en positions). Le rôle des dispositifs de traitement plus simples (opérateurs mathématiques ou booléens, traitement de signal, branchements conditionnels, animation) reste toutefois essentiel, car ils permettent d'affiner les techniques d'interaction prédéfinies et de construire des techniques répondant à des situations très spécifiques.

Les apports essentiels de notre approche

Le premier avantage de notre modèle d'interaction en entrée est qu'il permet de décrire toute la chaîne de gestion des entrées de manière explicite, du dispositif physique jusqu'à l'application. Dans les systèmes interactifs conventionnels, les techniques d'interaction sont concrètement implémentées sur plusieurs niveaux d'abstraction, la plupart de ces niveaux étant inaccessibles au programmeur d'applications ; celui-ci dispose en particulier d'une visibilité très limitée quant aux traitements effectués aux niveaux bas. À l'inverse, notre modèle expose tous les aspects de la gestion des entrées de manière homogène et modulaire, d'une manière qui permet de saisir des aspects habituellement obscurs de l'interaction en entrée : dans notre paradigme, les techniques d'interaction sont simplement des adaptateurs ou des compositions d'adaptateurs dont la disposition en série traduit des cascades de traitement de données et de feedback.

Une spécificité de notre approche est qu'elle repose sur un modèle d'exécution réactif. Les langages réactifs, dans lesquels la propagation de l'information est conceptuellement instantanée, sont particulièrement adaptés à la description de systèmes qui réagissent immédiatement à l'environnement. Aujourd'hui, seuls les aspects très bas-niveau de l'interaction sont gérés de façon réactive (pilotes de dispositifs, animation du pointeur), le reste étant pris en charge par des modèles à événements qui sont de type conversationnel (l'utilisateur attend que les événements soient traités). Nous pensons cependant que la majeure partie de l'interaction en entrée gagne à être décrite de façon réactive. Les aspects asynchrones et non-déterministes de l'interaction, liés par exemple à des techniques de reconnaissance vocale ou gestuelle, s'intègrent en outre dans notre modèle de façon très naturelle.

La métaphore de la connexion logicielle et le principe d'adaptateurs qui caractérisent notre modèle constituent des paradigmes naturellement appropriés à la description d'applications entièrement configurables en entrée. L'éditeur interactif d'ICON instrumente ces paradigmes avec des techniques d'interaction étudiées pour faciliter la construction de configurations d'entrée et offrir une configurabilité à un public plus large que les seuls développeurs. Les utilisateurs avancés peuvent ainsi, selon leur degré d'expertise, personnaliser des configurations d'entrée prédéfinies pour les adapter à leurs besoins ou à ceux d'autres utilisateurs. L'éditeur interactif d'ICON est également un outil de prototypage qui permet aux développeurs de construire et tester pour leur application un grand nombre de techniques de contrôle en fonction des dispositifs physiques potentiellement disponibles.

Les possibilités offertes par ICON et son éditeur interactif sont extrêmement nombreuses, et nous n'en n'avons exploré qu'un sous-ensemble. Nous avons décrit dans cette thèse plusieurs exemples de configurations fonctionnant avec des applications Swing déjà existantes ou avec une application de dessin conçue pour être exclusivement contrôlée avec ICON. Ces exemples illustrent diverses façons d'exploiter des dispositifs standard (clavier, souris) ou évolués (tablettes graphiques, reconnaissance vocale, dispositifs 3D) avec des adaptateurs prédéfinis (contrôle positionnel au clavier, interaction gestuelle, outils semi-transparents,…) ou par combinaison d'opérateurs simples (interaction bimanuelle, commandes vocales, techniques de bas niveau inédites), pour contrôler des applications interactives de façon générique ou dédiée.

Un avantage décisif de l'approche à flot de données est qu'elle est particulièrement adaptée à la description d'interactions Post-WIMP fortement concurrentes, directes et amodales. ICON encourage en outre l'emploi de dispositifs physiques multiples et la description de techniques d'interaction novatrices et dédiées à la tâche. De plus, ICON fournit les outils nécessaires pour décrire la majorité des paradigmes d'interaction non conventionnels que nous avons évoqués dans le premier chapitre. Les styles d'interaction pour lesquels ICON est moins adapté sont de deux types : d'abord, nous l'avons

vu, les comportements essentiellement orientés contrôle. Ensuite, les paradigmes Post-WIMP qui font un usage évolué de modalités naturelles : interprétations sémantiques avancées, fusion de modalités hétérogènes et prise en charge de l'ambiguïté. Nous qualifions ces dernières techniques de *communicationnelles* (l'interface est un interlocuteur intelligent) par opposition aux techniques *instrumentales* (l'interface est un ensemble d'outils) pour lesquelles ICON est le mieux adapté. Même des modèles à événements sophistiqués du type Multimodal Subarctic gagneraient cependant à reposer sur une infrastructure ICON pour décrire les niveaux bas de l'interaction en entrée.

Pour finir, ICON constitue une solution originale et efficace aux problèmes liés à l'adaptabilité en entrée. Il permet de décrire des applications hautement *contrôlables* grâce à une prise en charge de dispositifs d'entrée multiples, l'emploi d'abstractions de bas niveau pour décrire à la fois ces dispositifs et les besoins spécifiques de chaque application en termes d'entrées, la facilité de spécification d'interactions fortement concurrentes, et la présence d'une librairie extensible de techniques d'interaction Post-WIMP prêtes à l'emploi. Une *accessibilité* minimale est assurée par la présence d'un ensemble aisément extensible de techniques d'accessibilité génériques reposant sur le principe des adaptateurs, et peut être améliorée par la construction de techniques d'accessibilité dédiées à la tâche. L'éditeur interactif d'ICON permet en effet au développeur d'applications de répondre efficacement à un grand nombre de situations en termes d'entrées, qu'elles soient enrichies ou appauvries. Une fois construites, les configurations d'entrée sont hautement *configurables*, et peuvent être personnalisées à différents niveaux selon le degré d'expertise de l'utilisateur.

Exploitation d'ICON et perspectives

La boîte à outils en entrées ICON est distribuée sur Internet et est actuellement employée dans le cadre du projet GINA (Géométrie Interactive et NAturelle). Elle a d'abord été expérimentée sur une application de reconstruction interactive de scènes 3D à partir de photographies, et est actuellement utilisée dans le développement d'une application Post-WIMP de modélisation 3D exploitant les techniques de dessin d'architectes. ICON a également servi à monter une expérimentation visant à comparer l'utilisabilité de plusieurs techniques d'interaction. En outre, deux projets sont actuellement en cours pour une intégration totale d'ICON avec des boîtes à outils graphiques Post-WIMP.

Si ICON semble avoir aujourd'hui atteint la maturité nécessaire pour pouvoir être employé dans des projets de développement, notre travail demeure essentiellement exploratoire, et demande à être affiné, complété et généralisé. Une validation exhaustive de notre approche nécessiterait notamment des études d'utilisabilité portant sur l'éditeur interactif et la librairie de programmation ICON, la mise en œuvre d'applications plus nombreuses et plus complexes, et une formalisation de son modèle à flots de données réactif. Notre approche ouvre par ailleurs de nombreuses perspectives d'amélioration et d'extension. Nous avons par exemple évoqué des stratégies d'intégration avec des approches complémentaires (formalismes orientés-contrôle, modèles d'interaction communicationnels de haut niveau, modèles multicouches pour le feedback graphique), ainsi que des extensions possibles de notre modèle visant notamment à incorporer les caractéristiques pragmatiques des dispositifs physiques mis en évidence par Buxton.

Une perspective peut-être lointaine mais des plus intéressantes serait une généralisation de notre paradigme de connexion logicielle aux niveaux matériel et système d'exploitation, dans le but de tendre vers une nouvelle génération de plate-formes intégralement ouvertes en entrée. Selon nous, l'avènement du standard USB a rendu ce type d'évolution possible, notamment en fournissant un accès unifié et de niveau bas à un grand nombre de dispositifs d'entrée, et en autorisant les connexions et

les déconnexions à chaud. Nos postes de travail devraient non seulement autoriser les connexions dynamiques de dispositifs physiques divers et variés, mais devraient également permettre aux utilisateurs de spécifier leur emploi de façon simple et naturelle. Notre expérience avec ICON nous a convaincu de la faisabilité d'un tel système, et nous avons déjà évoqué un certain nombre de caractéristiques qui nous semblaient pertinentes pour une version grand public et allégée de notre configurateur, combinant dans des proportions raisonnables simplicité d'utilisation et pouvoir d'expression.

Remerciements

Merci à Jean-Daniel Fekete, pour ses qualités d'encadrant, à la fois scientifiques et humaines, pour la patience dont il a su faire preuve pendant ces nombreuses années où j'ai souvent douté, et pour avoir fini par me transmettre le virus de l'IHM. Il s'est beaucoup investi dans ces travaux, a été à l'origine de nombreuses idées, et ce sont véritablement les fruits d'un travail commun qui sont présentés dans ce mémoire. Je remercie aussi Gérard Hégron, pour sa patience également, et son soutien.

Merci à Stéphane Huot, pour m'avoir continuellement stimulé par son enthousiasme sans bornes et sa passion du travail en équipe, ainsi qu'aux autres thésards de l'équipe CMI : Frédéric Jourdan, Didier Boucard et Mohammad Ghoniem, avec lesquels je me suis senti très proche. Nul doute que je n'aurais pu venir à bout de ma thèse sans les encouragements et le soutien (et aussi la bonne humeur) de Stéphane, Frédéric et des autres. J'ai eu également grand plaisir à côtoyer les membres et ex-membres du département informatique de l'EMN, je pense en particulier à Geoffrey Subileau, Mathias Braux, Mohammed Tounsi, Abdallah, Philippe, Narendra, Rémi, Christine, Cédric et Christian, sans oublier Vanessa et les autres membres du CRITE. Merci aussi à Pierre Cointe pour m'avoir accordé sa confiance et offert son appui lors d'une période difficile.

Merci à ceux qui en montrant de l'intérêt pour mes travaux m'ont beaucoup encouragé, c'est-à-dire aux membres de mon jury de thèse, à Caroline Appert, et aux autres. Merci également à ceux de mes nouveaux collègues Toulousains du LIIHS, du CENA, et d'Intuilab, qui m'ont accompagné à la fin de mon calvaire et m'ont aidé lors de mes répétitions de soutenance.

Merci à mes proches pour avoir longtemps supporté mon irritabilité lorsqu'il fallait aborder le délicat sujet de l'avancement de ma thèse : mes parents et mon frère Christian, ainsi que Pascal, Nadir et Fatima, Paul et Lucie, Jérémy, Thomas, Stéphane et Florence. Merci aussi au personnel du Groupe Scolaire de la Maison d'Arrêt avec qui j'ai eu plaisir à travailler pendant la durée de mon service national : Jean-Marc Allain et tous les instituteurs spécialisés, les emplois-jeunes, Raphaël, ainsi que les détenus.

Merci tout particulièrement à Béatrice, ainsi qu'à sa proche famille : Jean, Jeanine, Philippe, Bernadette, Xavier, Thérèse, Laurent et Anita.

Annexe A

Structure d'une configuration d'entrée

Sommaire

Notre but dans ces trois annexes n'est pas d'introduire un nouveau formalisme data-flow, mais de décrire la structure et les mécanismes spécifiques aux configurations d'entrée de façon suffisamment univoque pour qu'ils puissent être compris et implémentés dans un langage impératif. Cette structure et ces mécanismes constituent le modèle ICOM *(Input Configuration Model)*. Dans la mesure du possible, nous essaierons de décrire ce modèle indépendamment de toute implémentation : les choix liés au style de programmation (hiérarchie des classes par exemple) sont laissés libres. Par ailleurs, nous emploierons le plus souvent des schémas graphiques génériques pour décrire les structures de données, indépendamment du langage visuel décrit dans le chapitre 3. Enfin, dans certains passages nécessitant des explications plus détaillées et à chaque fois que le langage naturel se révélera peu adapté à la description d'un mécanisme, des notations mathématiques simples seront employées.

Dans cette partie, nous introduisons les éléments de base du modèle ICOM, en décrivant leur structure et leurs relations. Cette partie concerne les *aspects statiques* des configurations d'entrée.

A.1 Vue d'ensemble

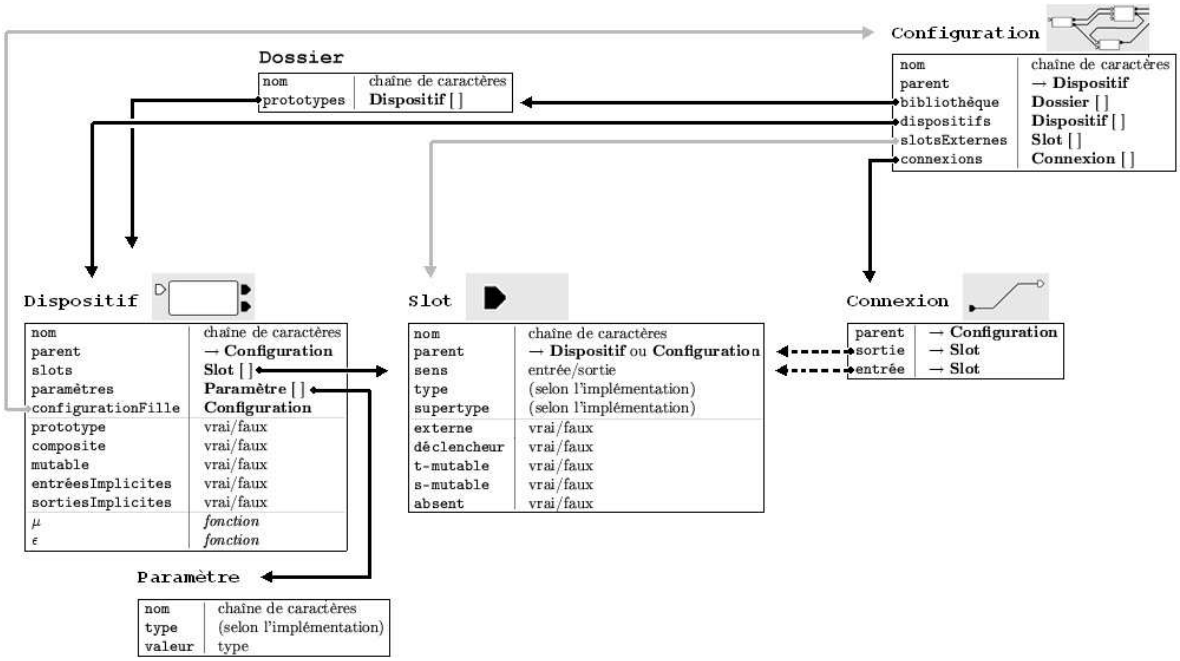


FIG. A.1 – Structure globale d'une configuration d'entrée, avec les briques de bases et leurs relations.

La figure A.1 fournit une vue globale de l'organisation structurelle d'une configuration d'entrée. Chaque entité est décrite par un ensemble d'attributs typés, c'est-à-dire une liste de noms (à gauche, en gras) et de types (à droite). Les flèches pleines décrivent les relations d'*agrégation* (X comporte un ou plusieurs Y) et les flèches en pointillés des relations de *référence* (X référence Y). Les flèches grisées traduisent des relations d'agrégation uniquement employées pour la composition et seront abordées plus tard.

La structure de base est la suivante : une configuration comporte un ensemble de *dispositifs* et

de *connexions*, ainsi que des *dossiers de dispositifs-prototypes*. Chaque dispositif comporte un ensemble de *slots* et de *paramètres*, et chaque *connexion* fait référence à deux slots. Un dispositif de type composite peut comporter une configuration dite *fille*, elle-même pouvant comporter des slots dits *externes*.

Dans la suite, nous décrivons les attributs des entités élémentaires de notre modèle dans l'ordre de la figure, de gauche à droite : le *dispositif*, les *paramètres*, le *dossier de prototypes*, le *slot*, la *connexion*, et la *configuration*.

A.2 Les quatre briques de base et leurs attributs

A.2.1 Dispositifs

La structure d'un dispositif est définie par un ensemble d'attributs et deux fonctions, qui sont énumérés dans le tableau A.2.1 puis décrits par la suite.

Dispositif	
`nom`	chaîne de caractères
`parent`	→ **Configuration**
`slots`	**Slot** [ ]
`paramètres`	**Paramètre** [ ]
`configurationFille`	**Configuration**
`prototype`	vrai/faux
`composite`	vrai/faux
`mutable`	vrai/faux
`entréesImplicites`	vrai/faux
`sortiesImplicites`	vrai/faux
μ	*fonction*
ε	*fonction*

TAB. A.1 – Structure d'un dispositif. La colonne de gauche liste le nom des attributs, celle de droite précise leur type.

- `nom :` Cet attribut contribue à identifier la fonction du dispositif, par exemple le type de traitement de données qu'il effectue. Il peut également refléter le rôle spécifique du dispositif au sein de la configuration.
- `parent :` Une référence à la configuration à laquelle appartient le dispositif, ou ou `null` s'il s'agit d'un dispositif-prototype.
- `slots :` Les *slots* d'un dispositif lui permettent de communiquer avec les autres dispositifs lors de la phase d'exécution (→ section A.2.2 page 199).
- `paramètres :` Un dispositif possède un ensemble éventuellement vide de *paramètres*. Les valeurs de ces paramètres déterminent certains aspects du comportement du dispositif pendant la phase de construction et d'exécution. Un dispositif ne peut pas comporter deux paramètres portant le même nom (→ section A.2.1 page suivante).
- `configurationFille :` La configuration-fille du dispositif, lorsqu'il s'agit d'un dispositif composite ; `null` sinon.

- `prototype :` Spécifie si le dispositif est un *dispositif-prototype* (→ section A.2.1).
- `composite :` Spécifie si le dispositif est de type *composite* (→ section A.4 page 204).
- `mutable :` Un dispositif est dit *mutable* s'il est susceptible de se spécialiser en fonction de la valeur prise par certains de ses attributs (→ annexe B).
- `entréesImplicites :` Indique si le dispositif reçoit des informations de l'environnement, c'est-à-dire des données autres que celles en provenance de ses slots d'entrée) (→ section C.5.1 page 232).
- `sortiesImplicites :` Indique si le dispositif émet des informations vers l'environnement, c'est-à-dire ailleurs que sur ses slots de sortie (→ section C.5.1 page 232).
- μ : μ est la *fonction de mutation* du dispositif, qui détermine le comportement de celui-ci pendant la phase de construction (→ annexe B).
- ε : ε est la *fonction d'exécution* du dispositif, qui détermine le comportement de celui-ci pendant la phase d'exécution (→ annexe C).

Paramètres

La structure d'un paramètre, décrite dans le tableau A.2.1, est simplement celle d'un attribut. Les paramètres peuvent être ainsi vus comme des attributs de dispositif supplémentaires.

Paramètre	
`nom`	chaîne de caractères
`type`	(selon l'implémentation)
`valeur`	type

TAB. A.2 – Structure d'un paramètre.

Dossiers de prototypes

Un dispositif-prototype est en tous points semblable à un dispositif, à ceci près qu'il ne fait pas partie d'une configuration, ne mute pas (il est obligatoirement dans un état consistant (→ section B.2.4 page 214)) et n'est jamais exécuté. Uniquement destiné à être dupliqué, celui-ci possède un état statique qui définit un paramétrage initial (valeur par défaut des paramètres) et un comportement (fonctions de mutation et d'exécution).

Les dispositifs-prototypes sont regroupés dans un dossier de prototypes dont la structure est la suivante :

Dossier	
`nom`	chaîne de caractères
`prototypes`	**Dispositif** [ ]

TAB. A.3 – Structure d'un dossier de prototypes.

A.2.2 Slots

Un *slot* est un état du dispositif qui est accessible aux autres dispositifs. Ces derniers ont la possibilité d'écrire sur le slot s'il s'agit d'un *slot d'entrée*, ou de lire sa valeur s'il s'agit d'un *slot de sortie* (figure 3.11 page 103). Ces échanges se font par l'intermédiaire des connexions (→ section A.3 page 201).

Les attributs d'un slot sont énumérés dans le tableau ci-dessous, et détaillés par la suite.

Slot	
`nom`	chaîne de caractères
`parent`	→ **Dispositif** ou **Configuration**
`sens`	entrée/sortie
`type`	(selon l'implémentation)
`supertype`	(selon l'implémentation)
`externe`	vrai/faux
`déclencheur`	vrai/faux
`t-mutable`	vrai/faux
`s-mutable`	vrai/faux
`absent`	vrai/faux

TAB. A.4 – Structure d'un slot.

- `nom :` Cet attribut identifie le slot au sein du dispositif. Le nom de chaque slot d'entrée d'un dispositif est *unique*, de même pour ses slots de sortie. Les noms comportant des points « . » définissent des slots structurés (structure cependant non explicite dans ICOM).
- `parent :` Une référence au dispositif ou à la configuration à laquelle appartient le slot, selon qu'il s'agisse d'un slot de dispositif ou d'un slot externe.
- `sens :` Cet attribut vaut `entrée` ou `sortie`, selon qu'il s'agisse d'un *slot d'entrée* ou d'un *slot de sortie*. Ce critère restreint l'ensemble des connexions possibles entre slots (→ section A.3 page 201).
- `type :` Le *type* d'un slot décrit le type de données qui y transitera lors de la phase d'exécution. Dans la phase de construction, celui-ci détermine simplement la *compatibilité* des connexions (→ section A.3.2 page 201).
- `supertype :` Le supertype d'un slot décrit l'ensemble des types que peut posséder un slot t-mutable. (→ annexe B)
- `externe :` Un slot *externe* est un slot isolé n'appartenant à aucun dispositif. Il sert à communiquer avec la configuration parente dans les dispositifs composites. Un slot non externe est également appelé *slot de dispositif*. (→ section A.4.1 page 204)
- `déclencheur :` Un slot est dit *déclencheur* s'il est susceptible de provoquer une mutation de son dispositif parent. Cet attribut vaut `faux` pour les slots absents. (→ annexe B)
- `t-mutable :` Un slot est dit *t-mutable* s'il est susceptible de changer de type. (→ annexe B)
- `s-mutable :` Un slot est dit *s-mutable* s'il est susceptible d'être ajouté ou supprimé du dispositif. Un slot ne peut pas être simultanément s-mutable et t-mutable. En outre, un slot déclencheur ne peut pas être s-mutable. (→ annexe B)
- `absent :` Spécifie si le slot est un slot *absent* ou un slot *présent*. Les slots présents sont créés par le dispositif, et les slots absents sont créés par les mécanismes de mutation. (→ annexe B)

A.2.3 Connexions

Une *connexion* est une relation liant deux slots, indiquant que ces deux slots partageront la même valeur lors de la phase d'exécution. La structure d'une connexion est décrite dans le tableau A.2.3.

Connexion	
`parent`	→ **Configuration**
`sortie`	→ **Slot**
`entrée`	→ **Slot**

TAB. A.5 – Structure d'une connexion.

- `parent :` Une référence à la configuration parente de la connexion.
- `sortie :` Une référence à un slot de sortie appartenant à `parent` ou à l'un de ses dispositifs-fils.
- `entrée :` Une référence à un slot d'entrée appartenant à `parent` ou à l'un de ses dispositifs-fils.

Les connexions sont orientées *de la sortie vers l'entrée*, et seront parfois notées **s** → **e**, *s* étant le slot de sortie et *e* le slot d'entrée.

A.2.4 Configuration d'entrée

Une configuration d'entrée est principalement constituée d'un ensemble de *dispositifs* et de *connexions*. Elle peut éventuellement être encapsulée dans un dispositif composite, auquel cas elle possède un dispositif parent. La structure d'une configuration d'entrée est décrite ci-dessous :

Configuration	
`nom`	chaîne de caractères
`parent`	→ **Dispositif**
`bibliothèque`	**Dossier** [ ]
`dispositifs`	**Dispositif** [ ]
`connexions`	**Connexion** [ ]
`slotsExternes`	**Slot** [ ]

TAB. A.6 – Structure d'une configuration d'entrée.

- `nom :` Ce nom contribue à identifier la fonction et le rôle de la configuration d'entrée.
- `parent :` Une référence au dispositif parent de la configuration, lorsque celle-ci est encapsulée dans un dispositif composite, `null` sinon.
- `bibliothèque :` La bibliothèque de dispositifs de la configuration, composée d'un ensemble de dossiers de prototypes.
- `dispositifs :` L'ensemble des dispositifs de la configuration d'entrée.
- `connexions :` L'ensemble des connexions qui relient les dispositifs entre eux. Les connexions seront abordées en détail dans la section suivante.
- `slotsExternes :` Une configuration peut comporter des slots isolés dits *externes*, qui lui permettent de communiquer avec l'extérieur. Ces objets sont décrits dans la section A.4 page 204.

A.3 Les connexions

A.3.1 Contraintes sur les connexions

Tout ensemble de connexions doit obéir aux trois contraintes suivantes :

Couplage : Une connexion appartenant à une configuration *C* peut relier soit :

1. Un slot de sortie appartenant à un dispositif de *C* à un slot d'entrée appartenant à un dispositif de *C* ;
2. Un slot de sortie appartenant à un dispositif de *C* à un slot d'entrée externe appartenant à *C* ;
3. Un slot de sortie externe appartenant à *C* à un slot d'entrée appartenant à un dispositif de *C* ;

En résumé, une connexion doit relier deux slots de sens opposés et être orientée du slot de sortie vers le slot d'entrée, et ne peut relier deux slots externes.

Unicité en entrée : Il existe au plus une connexion comportant un élément `entrée` donné. Ceci implique notamment que les connexions multiples sur un slot d'entrée sont interdits, ainsi que les connexions redondantes.

Acyclisme : Un ensemble de connexions ne doit pas générer de dépendances cycliques : pour toute configuration *C*, le *graphe de dépendances* (→ section A.3.5 page 204) de la *décomposition totale* (→ section A.4 page 204) de *C* doit être acyclique.

A.3.2 Types et sous-typage

Hormis les contraintes énumérés précédemment, toute connexion est autorisée indépendamment des types des slots. Cependant, notre modèle de connexion est typé et différencie les connexions *compatibles* des connexions *incompatibles*.

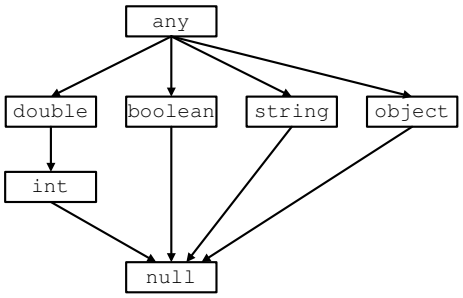


FIG. A.2 – Exemple de graphe de types pour les slots.

La figure A.2 représente le graphe de types que nous utilisons dans notre implémentation en Java, avec les relations de sous-typage. Les différents types de slots sont les `int` (nombres entiers), `boolean` (booléens), `double` (nombres flottants), `string` (chaînes de caractères), et `object` (objets). À cela s'ajoutent les types spéciaux `any` (qui représente tous les types possibles) et `null` (qui ne représente

aucun type). Ce graphe, comportant notamment le type `object`, est adapté à un langage à objets tel que Java.

Notre modèle ne nécessite pas la définition explicite d'un graphe de types. Au lieu de cela, il suppose qu'il existe un graphe de types avec à son sommet le type `any`, et à sa base le type `null`, et définit sur ce graphe les relations et opérations suivantes :

• Nous dirons que t_s est un ***sous-type*** de t (relation notée $t_s \subseteq t$) *ssi* t_s est successeur inclusif de t dans le graphe, c'est-à-dire ssi $t_s = t$ ou t_s est successeur direct ou indirect de t. De même, nous nommerons ***union*** de plusieurs types leur premier prédécesseur inclusif commun dans le graphe, et ***intersection*** de plusieurs types leur premier successeur inclusif commun dans ce même graphe. Les opérations d'union et d'intersection seront respectivement notées $\bigcup(t_1,\ldots,t_n)$ et $\bigcap(t_1,\ldots,t_n)$.

A.3.3 Attributs de connexions dérivés des slots

Nous nommons *attribut dérivé* un attribut d'un objet dont la valeur est fonction de celle des autres attributs de cet objet ou de ses objets accessibles (objets-fils ou objets référencés). Ils seront précédés d'un triangle ► . Ces attributs n'apportent pas d'information supplémentaire mais peuvent néanmoins servir à introduire des définitions utiles. Nous verrons également que lorsque la valeur de certains attributs est fixée dans un objet, d'autres attributs initialement libres peuvent devenir des attributs dérivés.

Les connexions comportent des caractéristiques de compatibilité, ainsi que d'autres attributs dérivés dont les valeurs sont fonction de leurs slots d'entrée et de sortie. Nous décrivons ces attributs, résumés dans le tableau A.3.3, en introduisant de nouveaux termes ainsi que leurs définitions.

Connexion	
`externe`	vrai/faux
`compatibilité`	compatible/incompatible
`compatibilitéStatique`	compatible/incompatible/mutable

TAB. A.7 – Attributs dérivés dans une connexion

► `externe :` Une *connexion externe* est une connexion reliant un slot de dispositif (slot pour lequel `externe=faux`) à un slot externe (slot pour lequel `externe=vrai`). Une connexion non externe, ou *connexion de dispositifs* est une connexion reliant deux slots de dispositifs. Une connexion ne peut relier deux slots externes.

► `compatibilité :` Une connexion est dite *compatible ssi* le type de son slot de sortie est un sous-type de celui de son slot d'entrée. Dans le cas contraire, elle est *incompatible*.

► `compatibilitéStatique :` Une connexion est dite *statiquement compatible ssi* le supertype de son slot de sortie est un sous-type de celui de son slot d'entrée, et que son slot d'entrée est ni s-mutable ni t-mutable. Une connexion est dite *statiquement incompatible ssi* le supertype de son slot de sortie n'est pas un sous-type de celui de son slot d'entrée, et que son slot de sortie est ni s-mutable ni t-mutable. Dans tous les autres cas, la connexion est dite *mutable*. Une connexion mutable est soit compatible soit incompatible, mais sa compatibilité peut changer suite à une mutation (→ annexe B). À l'inverse, une connexion statiquement compatible est toujours compatible, et une connexion statiquement incompatible est toujours incompatible.

La *compatibilité de types* n'entre pas en ligne de compte dans les contraintes de connexion vues précédemment car, comme nous le verrons dans l'annexe B, une connexion compatible peut très bien devenir incompatible au cours de l'édition et vice-versa. La *compatibilité statique* décrit cependant les évolutions possibles de cette compatibilité, et il est tout à fait possible (bien que nous ne le faisions pas) de refuser les connexions statiquement incompatibles.

A.3.4 Attributs de slots dérivés des connexions

De même que les slots induisent un certain nombre de propriétés sur les connexions, certaines propriétés caractérisent les slots en fonction des connexions existantes. Nous décrirons ces attributs dérivés tout en introduisant de nouveaux termes qui nous seront utiles dans les deux prochaines annexes. Ces attributs sont résumées dans le tableau A.3.4.

Slot	
`connexions`	**Connexion** [ ]
`slotsConnectés`	**Slot** [ ]
`connecté`	vrai/faux
`typeConnecté`	(selon l'implémentation)

TAB. A.8 – Attributs dérivés dans un slot

- `connexions :` Les *connexions d'un slot S* sont les connexions comportant *S* en entrée ou en sortie. Pour un slot d'entrée, cet attribut comporte au plus un élément.
- `slotsConnectés :` Les *slots connectés à S* sont les slots reliés à *S* par des connexions. Pour un slot d'entrée, cet attribut comporte au plus un élément.
- `connecté :` Un slot est dit *connecté* ssi il comporte au moins une connexion.
- `typeConnecté :` Le *type connecté* d'un slot de sortie est l'intersection des types de ses slots connectés, ou `any` s'il n'est pas connecté. Le *type connecté* d'un slot d'entrée est le type de son slot connecté, ou `null` s'il n'est pas connecté.

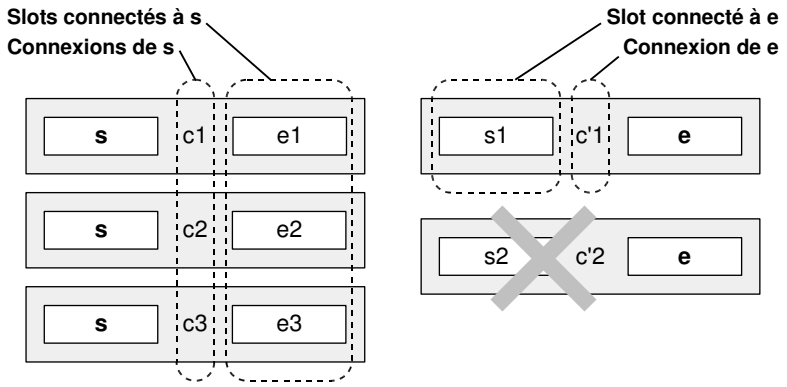


FIG. A.3 – Exemples de connexions : le slot de sortie *s* comporte trois connexions *c1*, *c2* et *c3* qui le relient aux slots d'entrée *e1*, *e2* et *e3*. Le slot d'entrée *e* comporte une connexion *c'1* qui le relie au slot de sortie *s1*. *e* ne peut comporter d'autre connexion.

A.3.5 Graphe de dépendances

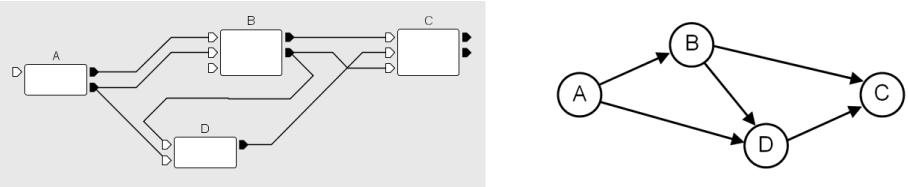


FIG. A.4 – Graphe de dépendances d'un ensemble de connexions.

Les connexions de dispositifs sont des arcs orientés reliant deux slots qui ont chacun un dispositif pour parent. Il est donc possible, à partir d'un ensemble de connexions de dispositifs, de construire un graphe orienté exprimant des dépendances entre les dispositifs (figure A.4). Ce graphe nous permet d'introduire les deux relations suivantes sur les dispositifs :

• Un dispositif D_1 est *successeur direct* d'un dispositif D_0 ssi il existe un arc $D_0 \rightarrow D_1$ dans le graphe de dépendances.

• Un dispositif D_1 est *successeur indirect* d'un dispositif D_0 ssi il existe un chemin qui va de D_0 à D_1 dans le graphe de dépendances. Cette relation est notée $D_0 \rightsquigarrow D_1$.

Dans les configurations ne comportant pas de dispositif composite, en particulier dans les configurations résultant d'une décomposition totale (→ section A.4), le graphe de dépendances est toujours un *graphe orienté acyclique*, ou DAG.

A.4 Composition de dispositifs

Les dispositifs sont *composables* : un ensemble de dispositifs interconnectés peut être décrit de façon équivalente par un dispositif unique appelé *dispositif composite*.

Un dispositif composite est essentiellement décrit par une configuration d'entrée, sa *configuration fille*. De façon générale, toute configuration d'entrée peut être encapsulée dans un dispositif composite, appelé *dispositif parent*. Ces deux relations de composition induisent également des relations hiérarchiques entre configurations et dispositifs : une configuration peut être *fille* ou *parente* d'une autre configuration du point de vue de la composition, de même que les dispositifs (figure A.5 page ci-contre). La communication d'un niveau à l'autre se fait par l'intermédiaire des *slots externes*.

A.4.1 Slots externes

Un *slot externe* est un type particulier de slot qui n'appartient pas à un dispositif mais à une configuration. Du point de vue de cette configuration, ce sont des points d'entrée et de sortie lui permettant d'échanger de l'information avec l'extérieur. Au niveau de la configuration parente, ils représentent les slots d'un dispositif composite : à chaque slot externe est associé un slot de sens opposé sur le dispositif composite (voir figure A.6 page suivante).

Un slot externe est uniquement défini par un *sens*, un *nom* et un *index*. Ses autres attributs sont

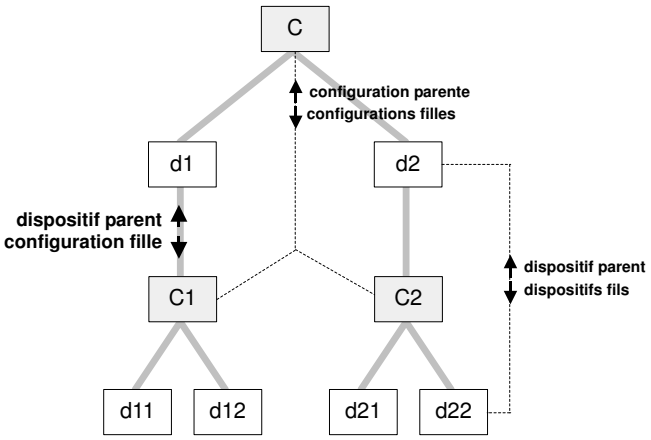


FIG. A.5 – Exemple de hiérarchies compositionnelles entre configurations et dispositifs : une configuration C comprend deux dispositifs composites $d1$ et $d2$ respectivement décrits par les configurations $C1$ et $C2$. Ces dernières comprennent chacun deux dispositifs.

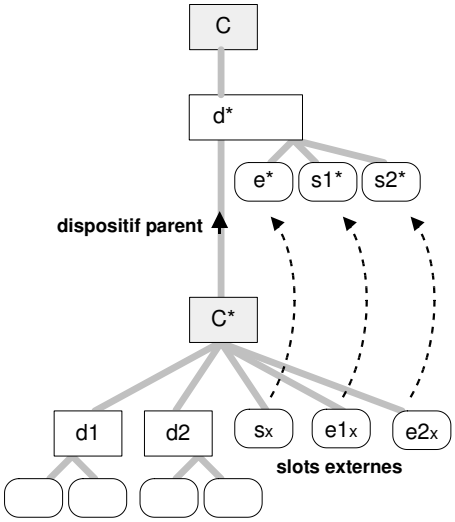


FIG. A.6 – La configuration $C^\star$ comprend trois slots externes s_x, $e1_x$ et $e2_x$ qui décrivent les slots de son dispositif parent d. À chaque slot d'entrée (resp. de sortie) externe est associé un slot de sortie (resp. d'entrée) sur le dispositif composite.

directement fonction des attributs de ses slots connectés (voir section suivante sur les connexions externes). Dans la liste qui suit, ces attributs dérivés sont précédés d'une flèche :

- `nom :` Cet attribut spécifie le nom du slot externe au sein de la configuration, ainsi que le nom du slot correspondant sur le dispositif composite. Dans une configuration d'entrée, les slots externes d'entrée ont des noms distincts, de même que les slots externes de sortie.
- `parent :` La configuration parente du slot composite.
- `sens :` Cet attribut vaut `entrée` ou `sortie`. Il spécifie si le slot externe est un slot externe *d'entrée* ou *de sortie*.
- ► `type :` Cet attribut a pour valeur le *type connecté* du slot externe (voir section suivante).
- ► `supertype :` Cet attribut a pour valeur `any`.
- ► `externe :` Cet attribut vaut `vrai`.
- ► `déclencheur :` Vaut `vrai` *ssi* il existe un slot connecté au slot externe pour lequel `declencheur=vrai`.
- ► `s-mutable :` Vaut `vrai` *ssi* il existe un slot connecté au slot externe pour lequel `s-mutable=vrai`.
- ► `t-mutable :` Vaut `vrai` *ssi* `s-mutable=faux` et il existe un slot connecté au slot externe pour lequel `t-mutable=vrai`..

A.4.2 Connexions externes

Comme nous l'avons vu précédemment, les connexions comportant un slot externe sont nommées *connexions externes*. Il existe deux types de connexions externes : les connexions externes *d'entrée* qui relient un slot d'entrée de dispositif à un slot externe de sortie, et les connexions externes *de sortie* qui relient un slot de sortie de dispositif à un slot externe d'entrée.

Les règles de connexion ainsi que les attributs et les relations définies dans la section A.3 page 201 sont les mêmes pour les connexions externes. Les contraintes d'unicité relatives aux connexions sur les slots d'entrée (section A.3 page 201) s'appliquent aux slots externes de sortie. Notons qu'une connexion externe est toujours compatible, car elle relie deux slots de même type.

A.4.3 Dispositifs composites

Un dispositif composite est entièrement défini par sa configuration fille. Hormis l'attribut `parent`, l'ensemble de ses attributs (précédés d'une flèche dans la liste ci-dessous) est fonction des attributs de cette configuration fille :

- `parent :` La configuration à laquelle appartient le dispositif, ou `null`.
- `configurationFille :` La configuration d'entrée qui décrit le dispositif composite.
- ► `nom :` Le nom d'un dispositif composite est le nom de sa configuration fille.
- ► `slots :` Les slots d'un dispositif composite correspondent aux slots externes *utilisés* dans la configuration fille, dans l'ordre indiqué par leur attribut `index`. Chaque slot du dispositif composite hérite des attributs du slot externe qui lui est associé.
- ► `paramètres :` Un dispositif composite ne comporte pas de paramètre.
- ► `composite :` Cet attribut vaut `vrai`.

► `prototype :` Cet attribut vaut `faux`.

► `mutable, entréesImplicites, sortiesImplicites :` Chacun de ces attributs vaut `vrai` *ssi* il est vrai pour au moins un dispositif dans la configuration fille.

A.4.4 Composition et décomposition

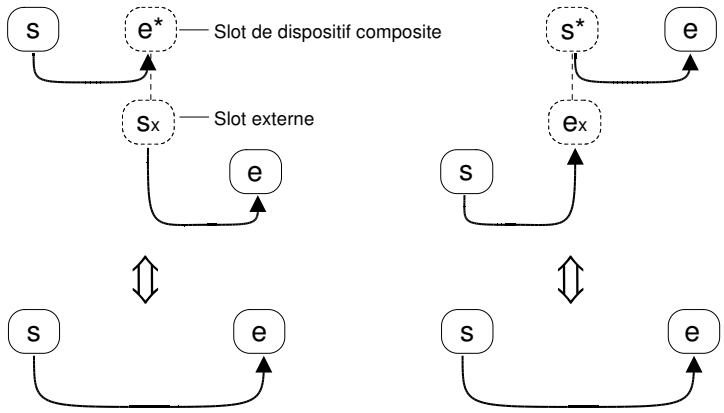


FIG. A.7 – Deux types de connexions inter-niveaux équivalentes à la connexion $s \rightarrow e$. À gauche, la connexion inter-niveaux *descendante* $(s \rightarrow e^{\star}, s_x \rightarrow e)$. À droite, la connexion inter-niveaux *ascendante* $(s \rightarrow e_x, s^{\star} \rightarrow e)$. Les slots intermédiaires permettent à la connexion $s \rightarrow e$ de remonter ou de redescendre d'un niveau dans l'arbre des configurations.

Nous appellerons *connexion inter-niveaux* toute paire de connexions reliant deux slots par l'intermédiaire d'un slot externe et de son slot de dispositif composite associé. Si cette connexion passe par le slot du dispositif composite puis par le slot externe, nous sommes dans le cas d'une connexion inter-niveaux *descendante* ; dans le cas contraire, il s'agit d'une connexion inter-niveaux *ascendante* (figure A.7). Dans les deux cas, la connexion inter-niveaux est équivalente du point de vue de l'exécution à une connexion directe entre le deux slots.

Des opérations de *composition* et de *décomposition* permettent de construire des structurations alternatives de l'arbre des configurations. La figure A.8 page suivante montre un exemple de composition ajoutant un niveau de configuration. Les dispositifs $d2$, $d3$ et $d4$ sont regroupés au sein d'un dispositif composite, qui peut ensuite être aplati pour redonner la configuration initiale.

La composition d'un ensemble de dispositifs Δ dans une configuration C s'obtient par l'algorithme A.1 page 209. L'opération inverse, consistant à décomposer un dispositif composite $d^{\star}$ dans C s'obtient par l'algorithme A.1 page 209. Ces algorithmes emploient la factorisation des connexions multiples sur un slot de sortie (figure A.9 page suivante).

Enfin, la *décomposition totale* d'une configuration C, utilisée pour le calcul des dépendances cycliques, s'obtient en appliquant récursivement l'algorithme A.1 page 209 sur l'ensemble des dispositifs de la configuration C.

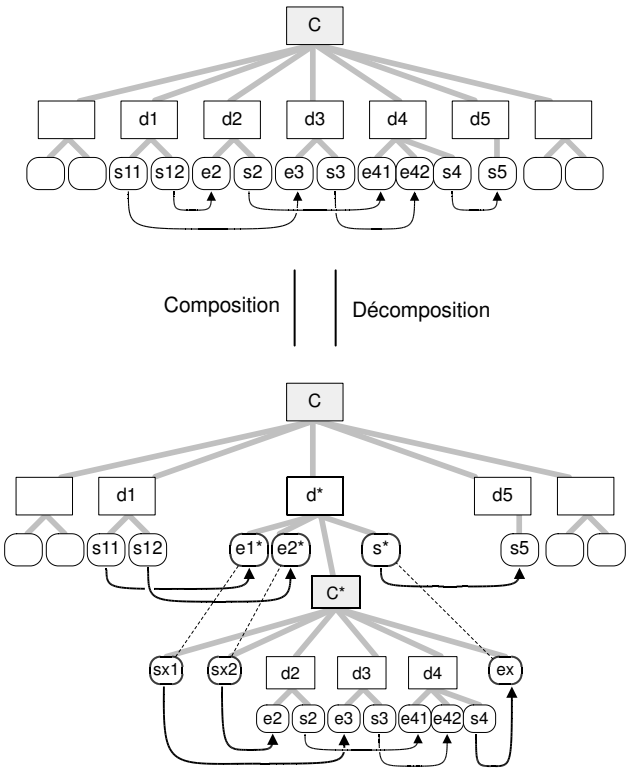


FIG. A.8 – Exemple de composition et de décomposition d'une configuration d'entrée. Les dispositifs $d1$, $d2$ et $d3$ sont déplacés dans la configuration-fille $C^\star$ et remplacés par le dispositif composite $d^\star$. Les connexions scindées telles que $s11 \rightarrow e3$ sont remplacées par des connexions inter-niveaux, tandis que les connexions internes telles que $(s2 \rightarrow e41)$ sont préservées (voir l'algorithme A.1). L'opération de décomposition supprime le dispositif composite et la configuration-fille, après avoir replacé les dispositifs-fils dans la configuration initiale et remplacé les connexions inter-niveaux par des connexions simples (voir l'algorithme A.2).

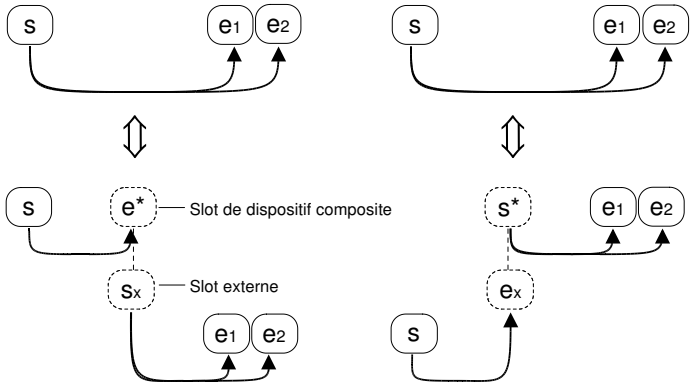


FIG. A.9 – Formes factorisées de connexions inter-niveaux multiples sur un slot de sortie.

1. Créer une configuration vide $C^\star$ contenant la même bibliothèque que C.
2. Créer un dispositif composite $d^\star$ de configuration-fille $C^\star$ et l'ajouter à C ;
3. Déplacer les dispositifs Δ vers $C^\star$;
4. Soit Σ l'ensemble des slots appartenant aux dispositifs Δ. Déplacer de C vers $C^\star$ toutes les connexions $s \rightarrow e$ de C vérifiant $s \in \Sigma$ et $e \in \Sigma$;
5. Pour toute connexion $s \rightarrow e$ de C vérifiant $s \notin \Sigma$ et $e \in \Sigma$:
   (a) s'il n'existe pas dans C de connexion $s \rightarrow e^\star$ avec $e^\star$ appartenant à $d^\star$:
   - ajouter un slot externe de sortie s_x dans $C^\star$, qui sera associé à un nouveau slot d'entrée $e^\star$ dans $d^\star$;
   - ajouter la connexion $s \rightarrow e^\star$ dans C ;
   - ajouter la connexion $s_x \rightarrow e$ dans $C^\star$.

   (b) s'il existe dans C une connexion $s \rightarrow e^\star$ avec $e^\star$ appartenant à $d^\star$:
   - ajouter la connexion $s_x \rightarrow e$ dans $C^\star$, s_x étant le slot externe associé à $e^\star$.
6. Pour toute connexion $s \rightarrow e$ de C vérifiant $s \in \Sigma$ et $e \notin \Sigma$:
   (a) s'il n'existe pas dans $C^\star$ de connexion $s \rightarrow e_x$ avec e_x slot externe :
   - ajouter un slot externe d'entrée e_x dans $C^\star$, qui sera associé à un nouveau slot de sortie $s^\star$ dans $D^\star$;
   - ajouter la connexion $s^\star \rightarrow e$ dans C ;
   - ajouter la connexion $s \rightarrow e_x$ dans $C^\star$.

   (b) s'il existe dans $C^\star$ une connexion $s \rightarrow e_x$ avec e_x slot externe :
   - ajouter la connexion $s^\star \rightarrow e$ dans C, $s^\star$ étant le slot de $d^\star$ associé à e_x.

Algorithme A.1: Composition des dispositifs Δ dans C.

A.4.5 Slots i-connectés

Il est utile d'introduire une définition alternative de la connexion, la *i-connexion*, qui ne prend en compte que les connexions effectives (éventuellement inter-niveaux) entre slots de dispositifs. Nous dirons ainsi qu'un slot de dispositif S' est *i-connecté* à un slot de dispositif S si S' est relié à S soit par une connexion, soit par une ou plusieurs connexions inter-niveaux successives.

Soit un slot S appartenant à un dispositif dans une configuration C. Soit C_0 la racine de l'arbre de configurations dont fait partie C.

• Le slot S' est *i-connecté* à S ssi S' est connecté à S dans la décomposition totale de C_0.

• *L'ensemble des slots i-connectés à S* est l'ensemble des slots connectés à S dans la décomposition totale de C_0.

1. Déplacer les dispositifs de $C^\star$ vers C ;
2. Déplacer toutes les connexions de dispositifs $s \rightarrow e$ de $C^\star$ vers C ;
3. Pour tout slot externe de sortie s_x de $C^\star$ et $e^\star$ son slot associé sur $d^\star$:
   (a) s'il n'existe pas de connexion $s \rightarrow e^\star$ dans C :
   - supprimer toutes les connexions $s_x \rightarrow e$ de $C^\star$.
   - supprimer s_x dans $C^\star$.

   (b) s'il existe une connexion $s \rightarrow e^\star$ dans C :
   - supprimer $s \rightarrow e^\star$ dans C ;
   - pour toute connexion $s_x \rightarrow e$ de $C^\star$:
     - supprimer $s_x \rightarrow e$ dans $C^\star$;
     - ajouter la connexion $s \rightarrow e$ à C ;
   - supprimer s_x dans $C^\star$.
4. Pour tout slot externe d'entrée e_x de $C^\star$ et $s^\star$ son slot associé sur $d^\star$:
   (a) s'il n'existe pas de connexion $s \rightarrow e_x$ dans $C^\star$:
   - supprimer toutes les connexions $s^\star \rightarrow e$ de C.
   - supprimer e_x dans $C^\star$.

   (b) s'il existe une connexion $s \rightarrow e_x$ dans $C^\star$:
   - supprimer $s \rightarrow e_x$ dans $C^\star$;
   - pour toute connexion $s^\star \rightarrow e$ de C :
     - supprimer $s^\star \rightarrow e$ de C ;
     - ajouter la connexion $s \rightarrow e$ à C.
   - supprimer e_x dans $C^\star$.
5. Supprimer le dispositif composite $d^\star$ et sa configuration-fille $C^\star$.

Algorithme A.2: Décomposition du dispositif $d^\star$ dans C.

Annexe B

Aspects dynamiques d'une configuration d'entrée

Sommaire

Dans cette partie, nous décrivons les principaux aspects dynamiques du modèle ICoM, qui traduisent le comportement d'une configuration d'entrée durant la phase de *construction* et d'*édition*. La plupart des notions abordées ici s'appuient sur les structures décrites dans l'annexe A.

B.1 Vue d'ensemble

L'essentiel du comportement en édition d'une configuration d'entrée repose sur les mécanismes de *mutation*. Les *dispositifs mutables* sont des dispositifs capables de se spécialiser ou se restructurer en modifiant la valeur de certains de leurs attributs en fonction de la valeur d'autres attributs, selon un mécanisme appelé *mutation*. Lors d'une mutation, le type de certains slots peut changer : ces slots sont nommés *t-mutables*. Une mutation peut également restructurer un dispositif en créant ou en supprimant des slots dynamiques, nommés *s-mutables*.

L'ensemble des attributs qui provoquent ces spécialisations et l'ensemble des attributs spécialisables constituent respectivement le *paramétrage* et le *m-paramétrage* du dispositif (figure B.1) :

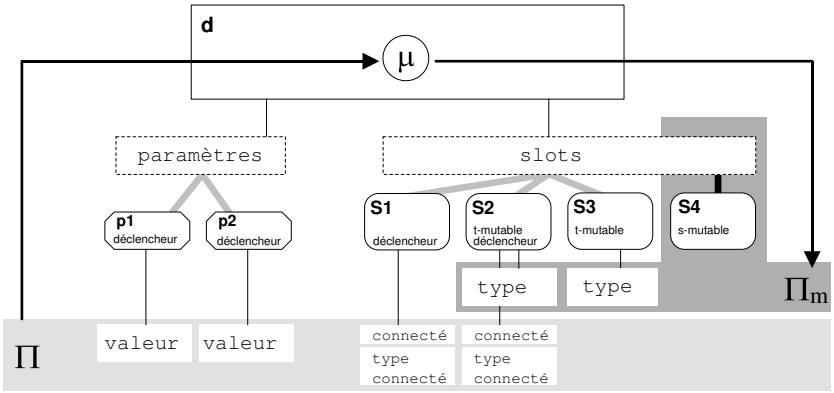


FIG. B.1 – Fonction de mutation μ d'un dispositif d comportant les paramètres $p1$, $p2$ et les slots $S1$, $S2$, $S3$ et $S4$. Π, la source de μ, est composé des valeurs prises par ses paramètres et des attributs de connexion de ses slots déclencheurs. Π_m, la cible de μ, décrit les types des slots t-mutables de d, ainsi que de l'ensemble des slots s-mutables que possède le dispositif après mutation.

1. Le *paramétrage* Π d'un dispositif décrit les valeurs prises par ses paramètres, et spécifie si ses slots déclencheurs sont connectés, ainsi que leurs types connectés.
2. Le *m-paramétrage* Π_m d'un dispositif décrit l'ensemble de ses slots s-mutables, ainsi que l'ensemble des types pris par ses slots t-mutables.
3. Chaque dispositif comporte une fonction de mutation $\mu : \Pi \mapsto \Pi_m$ qui à un paramétrage associe un m-paramétrage.

Après une introduction préliminaire sur les notions de signature et de valuation, nous définirons plus précisément Π, Π_m et la fonction μ, puis nous décrirons les mécanismes de base qui permettent d'appliquer et de propager les fonctions de mutation.

B.2 Définitions préliminaires

B.2.1 Types de slots

Un slot structurellement mutable ou *s-mutable* est un slot pour lequel l'attribut `s-mutable = vrai`. Un slot *t-mutable* est un slot pour lequel `t-mutable = vrai`, et un slot absent est un slot pour lequel `absent = vrai`.

Les slots s-mutables sont uniquement définis par leur `parent`, leur `nom`, leur `sens` et leur `type`. Les slots absents sont uniquement définis par leur `parent`, leur `nom`, et leur `sens`.

Un slot ne peut pas être simultanément s-mutable et t-mutable. En outre, *un slot déclencheur ne peut pas être s-mutable*.

B.2.2 Identifiants et valuations

Les définitions qui suivent s'appliquent à tout dispositif, qu'il soit mutable ou non.

• L'***identifiant*** $V(S)$ d'un slot S de dispositif est un couple formé de la valeur de ses attributs `nom` et `sens`, permettant de caractériser de façon unique ce slot sur son dispositif parent :

$V(S) = \langle S.nom, S.sens \rangle$

Tout couple de la forme $\langle nom, sens \rangle$ dont les éléments constitutifs ont pour types respectifs ceux des attributs de slots `nom` et `sens` est un *identifiant de slot*. Un identifiant de slot $\langle nom, sens \rangle$ ***désigne*** le slot S *ssi* $V(S) = \langle nom, sens \rangle$.

Soient $a1, \ldots, an$ des attributs de slots distincts, et différents des attributs `nom` et `sens`.

• La ***valuation*** $V_{a_1,\ldots,a_n}(S)$ d'un slot de dispositif S est un n-uplet comportant l'identifiant de S et les valeurs de ses attributs $a_1, \ldots, a_n$:

$V_{a_1,\ldots,a_n}(S) = \langle S.nom, S.sens, S.a_1, \ldots, S.a_n \rangle$

Tout n-uplet de la forme $\langle nom, sens, a_1, \ldots, a_n \rangle$, dont les éléments constitutifs ont pour types respectifs ceux des attributs de slots `nom`, `sens`, a_1, ..., a_n, est *une valuation de slot*, notée $V_{a_1,\ldots,a_n}$. Le couple $\langle nom, sens \rangle$ en constitue *l'identifiant*. Nous dirons qu'une valuation de slot *désigne* le slot S *ssi* son identifiant désigne le slot S.

De façon analogue, pour les paramètres :

• L'identifiant $V(P)$ d'un paramètre P est la valeur de son attribut `nom`, et la valuation $V_{valeur}(P)$ de ce paramètre est le couple formé par la valeur de ses attributs `nom` et `valeur`.

B.2.3 Paramétrages et m-paramétrages de dispositifs

Les définitions qui suivent s'appliquent à tout dispositif, qu'il soit mutable ou non.

• Le ***paramétrage*** d'un dispositif d est le couple :

$\Pi(d) = \langle V_{Pd}, V_{Sd} \rangle$

où V_{Pd} est l'ensemble des valuations V_{valeur} des *paramètres* de d :

$V_{Pd} = \{\langle P_1.nom, P_1.valeur\rangle, \ldots, \langle P_n.nom, P_n.valeur\rangle\}_{P_i \in d.parametres}$

et V_{Sd} est l'ensemble des valuations $V_{connecte,typeConnecte}$ des *slots déclencheurs* de d :

$V_{Sd} = \{\langle S_1.nom, S_1.sens, S_1.connecte, S_1.typeConnecte\rangle, \ldots,$
$\langle S_{n'}.nom, S_{n'}.sens, S_{n'}.connecte, S_{n'}.typeConnecte\rangle\}_{S_i \in d.slots\ et\ S_i.declencheur=vrai}$

• Le ***m-paramétrage*** d'un dispositif d est le couple :

$\Pi_m(d) = \langle V_{Ss}, V_{St}\rangle$

où V_{Ss} est l'ensemble de valuations V_{type} des slots *s-mutables* de d :

$V_{Ss} = \{\langle S_1.nom, S_1.sens, S_1.type\rangle, \ldots,$
$\langle S_{n''}.nom, S_{n''}.sens, S_{n''}.type\rangle\}_{S_i \in d.slots\ et\ S_i.s-mutable=vrai}$

et V_{St} est l'ensemble de valuations V_{type} des slots *t-mutables* de d :

$V_{Ss} = \{\langle S_1.nom, S_1.sens, S_1.type\rangle, \ldots,$
$\langle S_{n'''}.nom, S_{n'''}.sens, S_{n'''}.type\rangle\}_{S_i \in d.slots\ et\ S_i.t-mutable=vrai}$

• Un ***m-paramétrage valide*** pour un dispositif d est un couple de la forme :

$\Pi_m = \langle V_{Ss}, V_{St}\rangle$

où V_{Ss} est un ensemble de valuations V_{type} :

$V_{Ss} = \{\langle nom_1, sens_1, type_1\rangle, \ldots, \langle nom_{n''}, sens_{n''}, typen_{n''}\rangle\}$

dont les identifiants sont distincts et ne désignent pas de slot présent non s-mutable de d (slots pour lesquels $absent = faux$ et s-mutable$= faux$).

et V_{St} est un ensemble de valuations V_{type} :

$V_{St} = \{\langle nom_1, sens_1, type_1\rangle, \ldots, \langle nom_{n'''}, sens_{n'''}, typen_{n'''}\rangle\}$

dont les identifiants sont distincts et désignent tous des slots t-mutables de d. En outre, chaque élément $\langle nom, sens, type\rangle$ de V_{St} identifiant S vérifie $type \leqslant S.supertype$.

Notons qu'un slot s-mutable étant entièrement défini par ses attributs `parent`, `nom`, `sens` et `type`, un ensemble de valuations V_{type} suffit bien à décrire l'ensemble des slots s-mutables d'un dispositif.

B.2.4 Fonction de mutation et consistance

Les définitions qui suivent s'appliquent à tout dispositif, qu'il soit mutable ou non.

Soit d un dispositif, $\mathcal{P}$ l'ensemble de ses paramétrages possibles, et $\mathcal{P}_m$ l'ensemble de ses m-paramétrages valides.

Le dispositif d comporte une ***fonction de mutation*** μ, qui à chaque paramétrage Π de $\mathcal{P}$ associe un m-paramétrage Π_m de $\mathcal{P}_m$:

$$\mu : \left| \begin{array}{l} \mathcal{P} \rightarrow \mathcal{P}_m \\ \Pi \mapsto \Pi_m \end{array} \right.$$

• Un dispositif est dans un ***état consistant*** *ssi* il est non mutable, ou s'il satisfait à sa fonction de mutation :

$\Pi_m(d) = \mu(\Pi(d))$

B.3 Algorithmes

B.3.1 Mutation d'un dispositif

La *mutation* d'un dispositif consiste en la réévaluation des valeurs de ses attributs à la suite d'une modification de son paramétrage, dans le but d'assurer sa consistance. Cette réévaluation n'est nécessaire que dans la mesure où ce dispositif est mutable. Le mécanisme de mutation d'un dispositif est décrit par l'algorithme B.1.

- Soit $\Pi_m = \langle V_{Ss}, V_{St} \rangle$ l'image de $\Pi(d)$ par μ.
- Soit $\Sigma^{=}$ l'ensemble des slots s-mutables de d désignés par une des valuations de V_{Ss}.
- Soit Σ^{-} l'ensemble des slots s-mutables non connectés de d, non désignés dans V_{Ss}.
- Soit Σ_a^{+} l'ensemble des slots s-mutables connectés de d, non désignés dans V_{Ss}.
- Soit Σ_a^{-} l'ensemble des slots absents de d, désignés dans V_{Ss}.
- Soit V^{+} l'ensemble des valuations de V_{Ss} ne désignant aucun slot de d.
- Soit Σ_t l'ensemble des slots t-mutables de d désignés dans V_{St}.
- Soit Σ un ensemble de slots initialement vide.

1. Pour chaque slot S de $\Sigma^{=}$:
   - mettre à jour l'attribut *type* de S avec la valeur x de la valuation $\langle S.nom, S.sens, x \rangle$ de V_{Ss} ;
   - si la nouvelle valeur de $S.type$ est différente de l'ancienne, ajouter S à Σ.
2. Pour chaque slot S de Σ^{-} :
   - supprimer le slot S du dispositif d.
3. Pour chaque slot S de Σ_a^{+} :
   - mettre à jour les attributs de S :
     $S.s-mutable = faux$, $S.absent = vrai$, $S.type = null$ si $S.sens = entre$ et $S.type = any$ si $sens = sortie$;
   - si la nouvelle valeur de $S.type$ est différente de l'ancienne, ajouter S à Σ.
4. Pour chaque slot S de Σ_a^{-} :
   - mettre à jour les attributs de S :
     $S.s-mutable = vrai$, $S.absent = faux$, $S.type = x$ avec $\langle S.nom, S.sens, x \rangle$ valuation de V_{Ss} ;
   - si la nouvelle valeur de $S.type$ est différente de l'ancienne, ajouter S à Σ.
5. Pour chaque valuation $\langle x, y, z \rangle$ de V^{+} :
   - créer un nouveau slot s-mutable $S_{[parent=d,nom=x,sens=y,type=z]}$ et l'ajouter à d.
6. Pour chaque slot S de Σ_t :
   - mettre à jour l'attribut *type* de S avec la valeur x de la valuation $\langle S.nom, S.sens, x \rangle$ de V_{St} ;
   - si la nouvelle valeur de $S.type$ est différente de l'ancienne, ajouter S à Σ.

Algorithme B.1: Mutation d'un dispositif d de fonction de mutation μ, et construction de l'ensemble Σ des slots qui ont changé de type.

L'algorithme de mutation met à jour les slots du dispositif en fonction du m-paramétrage, en supprimant, en ajoutant, ou en mettant à jour le type des slots. L'emploi de slots *absents* évite la suppression de connexions à la suite d'une mutation. En particulier, un slot supprimé puis créé à nouveau conservera ses connexions.

L'algorithme distingue six cas possibles pour chaque slot : Les slots s-mutables de d qui sont absents du m-paramétrage sont supprimés s'ils ne sont pas connectés (slots Σ^-) et transformés en slots absents s'ils sont connectés (slots Σ_a^+). Les nouveaux slots s-mutables décrits par le m-paramétrage sont ajoutés s'ils n'existent pas en tant que slots absents (valuations V^+), et dérivés des slots absents dans le cas contraire (slots Σ_a^-). Enfin, l'algorithme met à jour les types des slots t-mutables (slots Σ_t) et des slots s-mutables qui sont présents à la fois sur le dispositif et dans le m-paramétrage (slots $\Sigma^=$). À la fin de l'algorithme, Σ comprend l'ensemble des slots ayant changé de type, qui servira à la propagation des mutations.

B.3.2 Propagation des mutations

La mutation d'un dispositif d est provoquée par une ou plusieurs des causes élémentaires suivantes :

1. la modification du type connecté d'un des slots déclencheurs de d ;
2. une connexion ou une déconnexion sur l'un des slots déclencheurs de d ;
3. la modification de la valeur d'un des paramètres de d.

En outre, la mutation de d a pour résultat une ou plusieurs conséquences élémentaires qui peuvent être classées en quatre catégories :

1. la modification du type d'un slot t-mutable ou s-mutable de d ;
2. la suppression d'un slot s-mutable connecté de d ;
3. la suppression d'un slot s-mutable non connecté de d ;
4. l'ajout d'un slot s-mutable à d.

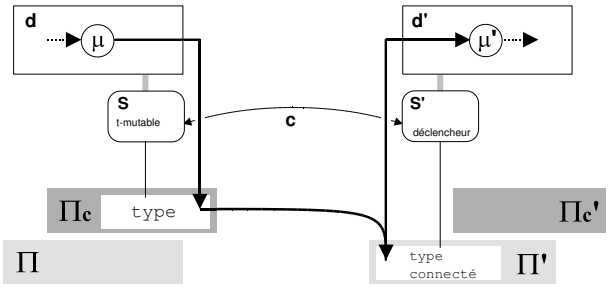


FIG. B.2 – Propagation d'une mutation : Le slot S, t-mutable, et le slot S', déclencheur, sont reliés par une connexion c de sens quelconque. À la suite d'une mutation du dispositif d, le type de S est modifié, ce qui a pour effet de modifier également le type connecté du slot S' et provoquer une mutation de d'.

Par l'intermédiaire des connexions, les conséquences de type 1 et 2 peuvent provoquer sur d'autres dispositifs des causes de type 1 et y déclencher des mutations : la mutation est alors *propagée* (figure B.2 page ci-contre). La mutation d'un dispositif est susceptible d'être propagée à tous les dispositifs directement connectés, *indépendamment du sens des connexions*. Ce mécanisme de propagation est décrit par l'algorithme B.2.

Répéter les étapes suivantes jusqu'à ce que Δ soit vide :

1. Copier Δ dans Δ' et effacer le contenu de Δ.
2. Pour tout dispositif d de Δ' :
   (a) Appliquer l'algorithme B.1 page 215 à d.
   Soit Σ l'ensemble des slots de d ayant changé de type.
   (b) Si $d \in \Delta$, retirer d de Δ.
   (c) Pour tout slot s appartenant à Σ, pour tout slot s' i-connecté à s :
   – Soit d' le dispositif parent de s'. Si s' est un slot déclencheur et si $d' \notin \Delta$, ajouter d' à Δ.

Algorithme B.2: Mutation avec propagation des dispositifs Δ dans une configuration C.

L'algorithme B.2 consiste à appliquer l'algorithme de mutation sur un ensemble de dispositifs, en se servant de la valeur de retour Σ pour déterminer l'ensemble des dispositifs vers lesquels les mutations doivent être propagées. L'algorithme est à nouveau appliqué sur ce dernier ensemble, et ainsi de suite jusqu'à ce qu'il n'y ait plus de dispositifs candidats. Notons que la propagation des mutations peut monter ou descendre dans l'arbre des configurations, à travers les slots i-connectés (voir section A.4.5 page 209).

Il est également important de noter que cet algorithme récursif opère en *largeur d'abord* plutôt qu'en *profondeur d'abord* : les dispositifs cibles de la propagation sont stockés au lieu d'être mutés immédiatement. Au niveau du slot, c'est l'algorithme de mutation (algorithme B.1 page 215) qui se charge du stockage des propagations. La propagation en largeur d'abord a pour avantage d'éviter des déclenchements inutiles de mutations (figure B.3).

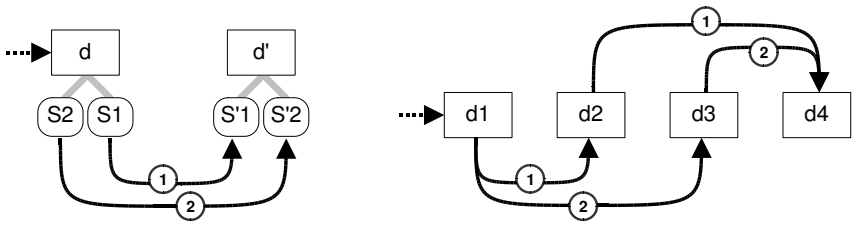


FIG. B.3 – Exemples de mutations inutiles. À gauche, une propagation en profondeur d'abord au niveau du slot : le dispositif d modifie son slot $S1$ puis son slot $S2$. La propagation instantanée des mutations déclenche deux mutations successives sur le dispositif d'. À droite, une propagation en profondeur d'abord au niveau du dispositif : le dispositif $d1$ déclenche une mutation sur $d2$ puis sur $d3$. Là aussi, la propagation instantanée déclenche deux mutations successives de $d4$.

B.3.3 Propagation bornée

Bien que les connexions d'une configuration d'entrée ne génèrent pas de dépendance cyclique, ce n'est pas le cas des mutations, qui se propagent indépendamment de l'orientation des connexions. Lorsque les mutations comportent des dépendances cycliques, l'algorithme B.2 page précédente peut ne pas se terminer, mais peut également converger.

L'algorithme B.3 est une variante de l'algorithme B.2 page précédente, qui se termine toujours. Le nombre de mutations sur chaque dispositif est limité à i_{max}. Lorsqu'un dispositif a muté i_{max} fois, celui-ci est considéré comme non stabilisé. À terme, l'algorithme fournit dans Δ_{NS} l'ensemble des dispositifs non stabilisés. Si cet ensemble est non vide, la propagation est considérée comme non stabilisée dans son ensemble.

De par sa propriété de terminaison, l'algorithme B.3 est de loin préférable à l'algorithme B.2 page précédente. Deux stratégies sont possibles pour l'exploiter : soit interdire les connexions qui induisent une propagation non stabilisée, soit autoriser provisoirement (hors exécution) les dispositifs non stabilisés dans une configuration. Un éditeur interactif pourrait par exemple mettre en évidence les « erreurs » dans l'espoir que l'utilisateur les corrige. Une manière de stabiliser la partie problématique de la configuration consiste à insérer des dispositifs « typeurs », dont les types en entrée et en sortie sont fixés par paramétrage : cela équivaut à ajouter des contraintes à un problème sous-contraint.

- Soit $i : \Delta'' \rightarrow \mathbb{N}$ une fonction qui associe un entier à des dispositifs, et dont l'ensemble de départ est initialement vide et Δ_{NS} un ensemble de dispositifs initialement vide également.
- Répéter les étapes suivantes jusqu'à ce que Δ soit vide :
  1. Copier Δ dans Δ' et effacer le contenu de Δ.
  2. Pour tout dispositif d de Δ' :
     (a) Si d ne fait pas partie de l'ensemble de départ de i, ajouter $d \mapsto i_{max}$ à i.
     (b) Si $i(d) = 0$, ajouter d à Δ_{NS}. Sinon :
        i. Appliquer l'algorithme B.1 page 215 à d.
        Soit Σ l'ensemble des slots de d ayant changé de type.
        ii. Décrémenter $i(d)$ dans i.
        iii. Si $d \in \Delta$, retirer d de Δ.
        iv. Pour tout slot s appartenant à Σ, pour tout slot s' i-connecté à s :
        – Soit d' le dispositif parent de s'. Si s' est un slot déclencheur et si $d' \notin \Delta$, ajouter d' à Δ.

Algorithme B.3: Mutation avec propagation bornée à i_{max} itérations des dispositifs Δ dans une configuration C. Cet algorithme construit également l'ensemble Δ_{NS} des dispositifs non stabilisés.

B.3.4 Exemples de fonctions de mutation

Le système de mutations, très général, peut se décliner en un certain nombre de mécanismes de spécialisation concrets. Parmi ceux-ci, nous donnerons un exemple de typage, de spécialisation structurelle par paramétrage, et d'adaptation bi-directionnelle. D'autres exemples peuvent être donnés, tels que la spécialisation structurelle par connexion, ou le typage par paramétrage.

Exemple de typage

Voici un exemple de dispositif dont les slots prennent le type `int` lorsque les slots d'entrée sont connectés à des entiers et le type `double` dans les autres cas :

Soit un dispositif mutable *add* ayant deux slots d'entrée nommés *in*1 et *in*2, un slot de sortie nommé *out*, et ne comportant pas de paramètre. *in*1, *in*2 et *out* sont t-mutables et possèdent comme supertype `double`. Les slots déclencheurs sont *in*1 et *in*2. La fonction de mutation μ_{add} de ce dispositif est la suivante :

$$\mu_{add} : \left| \begin{array}{l} \mathcal{P} \rightarrow \mathcal{P}_m \\ \Big\langle \varnothing, \big\{ \langle in1, entrée, c_1, x_1 \rangle, \langle in2, entrée, c_2, x_2 \rangle \big\} \Big\rangle \mapsto \\ \quad \Big\langle \varnothing, \big\{ \langle in1, entrée, y \rangle, \langle in2, entrée, y \rangle, \langle out, sortie, y \rangle \big\} \Big\rangle \end{array} \right.$$

avec :

$$\left| \begin{array}{l} y = int \text{ si } x_1 = int \text{ et } x_2 = int \\ y = double \text{ sinon.} \end{array} \right.$$

Exemple de spécialisation structurelle

Voici un exemple de dispositif dont le nombre de slots de sortie et d'entrée est paramétrable :

Soit un dispositif mutable *multipass* comportant deux paramètres respectivement nommés *inCount* et *outCount*. Le dispositif ne comporte pas d'autre slot que les slots s-mutables générés par la fonction de mutation. La fonction de mutation $\mu_{multipass}$ de ce dispositif est la suivante :

$$\mu_{multipass} : \left| \begin{array}{l} \mathcal{P} \rightarrow \mathcal{P}_m \\ \Big\langle \big\{ \langle inCount, x_1 \rangle, \langle outCount, x_2 \rangle \big\}, \varnothing \Big\rangle \mapsto \big\langle V_{Ss}^1 \cup V_{Ss}^2, \varnothing \big\rangle \end{array} \right.$$

avec :

$$\left| \begin{array}{l} V_{Ss}^1 = \varnothing \quad \text{si } x_1 \leq 0, \\ V_{Ss}^1 = \big\{ \langle in1, entrée, int \rangle, \ldots, \langle in_{x_1}, entrée, int \rangle \big\} \quad \text{sinon.} \end{array} \right.$$

$$\left| \begin{array}{l} V_{Ss}^2 = \varnothing \quad \text{si } x_2 \leq 0, \\ V_{Ss}^2 = \big\{ \langle out1, sortie, int \rangle, \ldots, \langle out_{x_2}, sortie, int \rangle \big\} \quad \text{sinon.} \end{array} \right.$$

Exemple d'adaptation bi-directionnelle

Enfin, voici un exemple de dispositif de type « adaptateur », dont les types s'adaptent en entrée et en sortie :

Soit un dispositif mutable *autocast* comportant un slot d'entrée et un slot de sortie respectivement nommés *in* et *out*, et ne comportant pas de paramètre. *in* et *out* sont déclencheurs et t-mutables, et possèdent comme supertype `any`. La fonction de mutation $\mu_{autocast}$ de ce dispositif est la suivante :

$$\mu_{autocast} : \left| \begin{array}{l} \mathcal{P} \rightarrow \mathcal{P}_m \\ \Big\langle \varnothing, \big\{ \langle in, entrée, c_1, x_1 \rangle, \langle out, sortie, c_2, x_2 \rangle \big\} \Big\rangle \mapsto \\ \quad \Big\langle \varnothing, \big\{ \langle in, entrée, x_1 \rangle, \langle out, sortie, x_2 \rangle \big\} \Big\rangle \end{array} \right.$$

B.4 Opérations sur les configurations

La structure d'une configuration d'entrée est susceptible d'évoluer dans le temps : elle peut être éditée. Cette évolution est réglée par un ensemble déterminé d'*opérations*, qui décrivent les évolutions possibles d'une configuration tout en imposant des contraintes sur celles-ci.

B.4.1 Opérations élémentaires

Une *opération élémentaire* sur une configuration est une fonction qui à une configuration C et un ensemble de dispositifs Δ associe une nouvelle configuration C' et un nouvel ensemble Δ'. Δ' comprend les dispositifs de Δ plus les dispositifs susceptibles d'être non consistants dans C', c'est-à-dire les dispositifs pouvant nécessiter une mutation.

Quatre opérations élémentaires sont énumérés par la suite, et décrits chacun par un algorithme simple montrant comment C' et Δ' s'obtiennent à partir de C et Δ.

Clonage de dispositifs

Le clonage d'un dispositif non composite d dans une configuration consiste à créer puis à ajouter dans cette configuration un nouveau dispositif possédant les mêmes valeurs d'attributs, mais ne comportant pas de connexion.

$Op_{cloner(d)} : (C, \Delta) \mapsto (C', \Delta')$

avec $d \in C.dispositifs$.

1. Création d'un dispositif d' vide ;
2. Copie dans d' des valeurs des attributs atomiques et des fonctions de d ;
3. Copie dans d' des paramètres et des slots non s-mutables et non absents de d ;
4. Ajout de d' à $C.dispositifs$;
5. Ajout de d' à Δ.

Ajout d'un dispositif

L'ajout d'un nouveau dispositif dans une configuration consiste en la copie d'un des dispositifs prototypes présents dans un des dossiers de prototypes de la configuration. L'opération d'ajout est la même que celle de clonage (même algorithme), à ceci près que le dispositif cloné appartient à l'un des dossiers de prototypes de la configuration.

$Op_{ajouter(p)} : (C, \Delta) \mapsto (C', \Delta')$

avec $p \in D.prototypes$ et $D \in C.bibliotheque$.

1. Appliquer $Op_{cloner(p)}$ à (C, Δ) ;
2. $p.parent = C$;

Connexion

Une opération de connexion consiste à relier un slot de sortie à un slot d'entrée par une nouvelle connexion. Elle est définie ainsi pour deux slots non externes :

$Op_{connecter(s,e)} : (C,\Delta) \mapsto (C',\Delta')$

avec :

- $s.parent \in C.dispositifs$
- $e.parent \in C.dispositifs$
- $s \rightarrow e$ obéit aux règles de couplage, d'unicité et d'acyclisme (→ section A.3 page 201).

1. Création d'une nouvelle connexion $x = s \rightarrow e$;
2. Ajout de x à $C.connexions$;
3. Ajout de $\{d_1, d_2\}$ à Δ.

Déconnexion

Une opération de déconnexion entre deux slots consiste à supprimer la connexion correspondante dans la configuration.

$Op_{deconnecter(s,e)} : (C,\Delta) \mapsto (C',\Delta')$

avec :

- $s.parent \in C.dispositifs$
- $e.parent \in C.dispositifs$
- $s \rightarrow e \in C.connexions$.

1. Suppression de la connexion $s \rightarrow e$ de $C.connexions$;
2. Ajout de $\{d_1, d_2\}$ à Δ.

Suppression d'un dispositif

La suppression d'un dispositif dans une configuration implique la suppression de toutes les connexions qui lui sont liées.

$Op_{supprimer(d)} : (C,\Delta) \mapsto (C',\Delta')$

avec $d \in C.dispositifs$

1. Pour toute connexion $s \rightarrow e$ de C telle que $s \in d.slots$ ou $e \in d.slots$, appliquer $Op_{deconnecter(s,e)}$ à (C,Δ) ;
2. Suppression de d dans $C.dispositifs$.

B.4.2 Opérations consistantes

Une *opération consistante* sur une configuration est une fonction qui à une configuration C associe une nouvelle configuration C' dont les dispositifs sont consistants. Une opération consistante peut être définie à partir d'une suite d'opérations élémentaires, de la manière suivante :

$Op_{(Op_1,\ldots,Op_n)} : C \mapsto C'$

1. Soit $\Delta = \varnothing$;
2. Appliquer dans l'ordre les Op_i sur (C, Δ) ;
3. Appliquer l'algorithme de mutation avec propagation sur Δ.

Annexe C

Exécution d'une configuration d'entrée

Sommaire

C.1 Introduction

Nous décrivons ici la partie exécution du modèle ICOM, c'est-à-dire le comportement des configurations d'entrée lors de la phase de lancement et d'exécution. Notre algorithme d'exécution est de type *réactif* ce qui signifie que chaque modification des entrées est répercutée et propagée dans la configuration en un temps conceptuellement nul (voir 3.3 page 99). La plupart des notions abordées ici s'appuient sur les structures décrites dans l'annexe A.

C.2 Définitions préalables

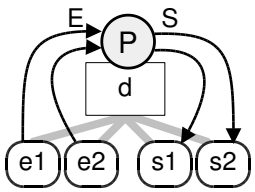


FIG. C.1 – Schéma d'un processeur.

Le comportement en exécution d'une configuration d'entrée est essentiellement décrit par des *processeurs*. Un processeur est une fonction qui aux valeurs prises par les slots d'entrée associe des valeurs aux slots de sortie (figure C.1). Lorsqu'une configuration est lancée, des structures sont allouées pour stocker la valeur de chaque slot, et chaque dispositif crée un processeur en fonction de son paramétrage, selon sa *fonction d'exécution* ε.

Les valeurs manipulées par les processeurs sont des valeurs de variables classiques associées à un booléen appelé *signal* : il s'agit de *signaux valués*. Un signal valué est propagé aux autres processeurs seulement si le signal est présent (booléen à vrai). Des modifications de la valeur d'une variable sont nécessairement propagées, mais une valeur non modifiée peut également être propagée. Un signal valué peut par conséquent représenter aussi bien un état qu'un événement.

Dans cette section, nous introduisons les définitions relatives aux signaux valués, aux processeurs, et aux fonctions d'exécution.

C.2.1 Signaux valués

Une *variable* est définie par ensemble appelé type et un élément de cet ensemble appelé valeur :

$V = \langle X, x \rangle, x \in X$

Un *signal valué* est la combinaison d'une variable et d'un booléen appelé *signal* :

$S = \langle V, s \rangle = \big\langle \langle X, x \rangle, s \big\rangle, x \in X, s \in \{vrai, faux\}$

La *valeur* d'un signal valué est un couple formé par la valeur x de sa variable V (appelée *valeur de variable* de S) et la valeur de son signal :

$v = \langle x, s \rangle$

L'ensemble des *valeurs possibles* de S, noté $\mathcal{V}(S)$, est $\mathcal{V}(S) = \{v\}_{v \in X \times \{vrai, faux\}}$.

C.2.2 Historiques

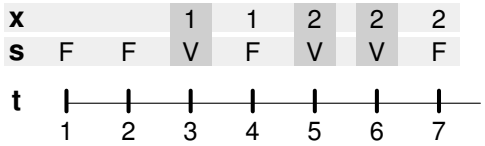


FIG. C.2 – Évolution possible d'un signal valué au cours du temps.

La valeur d'un signal valué évolue au cours du temps, que nous supposons discret et représenté par un entier strictement positif. Nous noterons $v^t = \langle x^t, s^t \rangle$ la valeur d'un signal valué à l'instant $t \in \mathbb{N}^*$.

Un signal valué évolue en accord avec la condition suivante :

$\forall t \in \mathbb{N}^*, x^{t+1} \neq x^t \Rightarrow s^{t+1} = vrai$

Autrement dit, le changement de la valeur de variable est une condition suffisante (mais non nécessaire) pour que le signal soit à vrai. En outre, un signal valué est considéré comme *n'ayant pas de valeur de variable* tant que son signal n'a pas reçu la valeur vrai au moins une fois. La figure C.2 illustre une évolution possible d'un signal valué au cours du temps, obéissant à ces deux conditions.

Un *historique* d'un signal valué S à l'instant t, noté $h^t(S)$, décrit une évolution de sa valeur depuis $t = 1$ jusqu'à t strictement positif :

$h^t(S) = \langle v^1, \ldots, v^t \rangle$

La condition de changement de valeur de variable énoncée plus haut est traduite par la définition de l'ensemble des historiques possibles $\mathcal{H}(S)$. La condition d'absence de valeur de variable est pour sa part traduite par la définition de la relation d'équivalence $\equiv$ entre deux historiques de $\mathcal{H}(S)$.

L'ensemble des *historiques possibles* $\mathcal{H}(S)$ d'un signal valué S est :

$\mathcal{H}(S) = \left\{ \langle v^1, \ldots, v^t \rangle \right\}_{t \in \mathbb{N}^*, v^i \in X_i \times \{vrai, faux\}, x^{i+1} \neq x^i \Rightarrow s^{i+1} = vrai}$

La *relation d'équivalence* $\equiv$ entre deux historiques de $\mathcal{H}(S)$ est définie comme suit :

Soient h et $h' \in \mathcal{H}(S)$.
$h \equiv h'$ ssi
$h = h'$ ou si h et h' sont de la forme :
$h = \left\langle \langle x, faux \rangle, \ldots, \langle x, faux \rangle, v^k, \ldots, v^t \right\rangle$
$h' = \left\langle \langle x', faux \rangle, \ldots, \langle x', faux \rangle, v^k, \ldots, v^t \right\rangle$

C.2.3 Signaux valués multiples

Par commodité pour la suite, nous étendrons les définitions précédentes aux signaux valués multiples du type $S = \langle S_1, \ldots, S_n \rangle$:

Les *valeurs* de S sont de la forme $v = \langle v_1, \ldots, v_n \rangle$. L'ensemble des *valeurs possibles* de S est $\mathcal{V}(S) = \left\{ \langle v_1, \ldots, v_n \rangle \right\}_{v_i \in X_i \times \{vrai, faux\}}$

Les *historiques* $h^t(S)$ de S sont de la forme :

$h^t(S) = \langle h^t(v_1), \ldots, h^t(v_n) \rangle = \left\langle \langle v_1^1, \ldots, v_1^t \rangle, \ldots, \langle v_n^1, \ldots, v_n^t \rangle \right\rangle$

L'ensemble des *historiques possibles* $\mathcal{H}(S)$ de S est :

$\mathcal{H}(S) = \left\{ \left\langle \langle v_1^1, \ldots, v_1^n \rangle, \ldots, \langle v_n^1, \ldots, v_n^t \rangle \right\rangle \right\}_{t \in \mathbb{N}^*, v_i^j \in X_i \times \{vrai, faux\}, x_i^{j+1} \neq x_i^j \Rightarrow s_i^{j+1} = vrai}$

La *relation d'équivalence* $\equiv$ entre deux historiques de $\mathcal{H}(S)$ est définie comme suit :

Soient h et $h' \in \mathcal{H}(S)$, avec $h = \langle h_1, \ldots, h_n \rangle$ et $h' = \langle h'_1, \ldots, h'_n \rangle$.
$h \equiv h'$ ssi $\forall i \in [1, n]$, $h_i \equiv h'_i$

C.2.4 Processeurs

Un *processeur* est une fonction qui à chaque historique d'un ensemble de signaux valués nommés *signaux d'entrée* associe des valeurs à d'autres signaux valués, nommés *signaux de sortie*.

Soit $P_{E,S}$ un processeur défini sur les signaux d'entrée $E = \langle e_1, \ldots, e_m \rangle$ et les signaux de sortie $S = \langle s_1, \ldots, s_n \rangle$. La fonction $P_{E,S}$ est définie comme suit :

$$P_{E,S} : \left| \begin{array}{l} \mathcal{H}(E) \rightarrow \mathcal{V}(S) \\ h^t(E) \mapsto v(S) \end{array} \right.$$

Cette fonction vérifie en outre la condition d'absence de valeur énoncée précédemment, c'est-à-dire qu'elle « ne peut pas lire » les variables qui n'ont pas encore reçu de signal. Cette condition s'énonce ainsi :

$\forall h$ et $h' \in \mathcal{H}(E)$, $h \equiv h' \Rightarrow P_{E,S}(h) = P_{E,S}(h')$

Un processeur $P_{E,S}$ est dit *passif* si en plus, il ne peut pas générer de signal (et à fortiori de valeur) sans en recevoir, ce qui se traduit par la condition suivante :

$\forall h \in \mathcal{H}(E)$
tel que $h = \left\langle \left\langle v_1^1, \ldots, v_1^{t-1}, \langle x_1^t, faux \rangle \right\rangle, \ldots, \left\langle v_n^1, \ldots, v_n^{t-1}, \langle x_n^t, faux \rangle \right\rangle \right\rangle$
l'image de h est de la forme $P_{E,S}(h) = \left\langle \langle y_1, faux \rangle, \ldots, \langle y_m, faux \rangle \right\rangle$

Dans le cas contraire, le processeur est dit *actif*.

C.2.5 Fonction d'exécution d'un dispositif

Comme nous l'avons vu en guise d'introduction, le comportement en exécution d'un dispositif est décrit par un processeur qui opère sur les signaux d'entrée et de sortie associés à ses slots d'entrée et de sortie (ces signaux valués sont créés lors du lancement de la configuration, nous le verrons dans la section suivante).

Le comportement en exécution dépend uniquement des valeurs prises par les paramètres du dispositif, ainsi que de ses types connectés s'il est mutable. À chaque *paramétrage* (→ section B.2.3

page 213) d'un dispositif correspond par conséquent un processeur. La correspondance paramétrage/processeur est décrite par la *fonction d'exécution* ε du dispositif, qui à chacun de ses paramétrages possibles associe un processeur P :

$$\varepsilon : \Pi(d) \mapsto P$$

C.3 Lancement et exécution d'une configuration

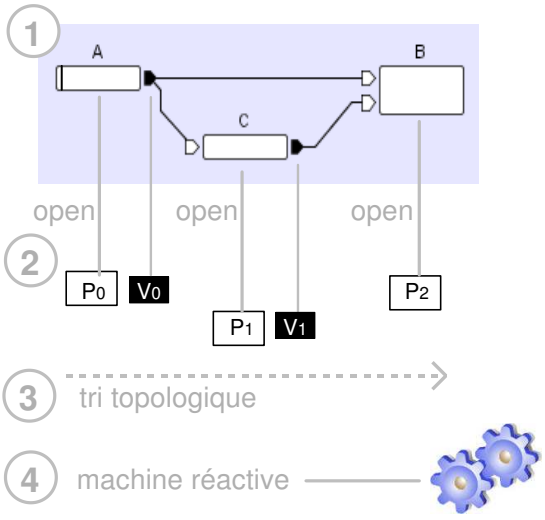


FIG. C.3 – Les quatre phases du processus de lancement d'une configuration.

Le *lancement* d'une configuration d'entrée consiste en un ensemble d'opérations destinées à préparer cette configuration d'entrée à l'exécution. Ces mécanismes aboutissent à la création d'une *machine réactive*, qui sera la version exécutable de la configuration d'entrée. Les différentes étapes de ce processus, illustrées sur la figure C.3 pour une configuration-exemple, sont les suivantes :

1. Aplatissement de l'arbre des configurations
2. Création des valeurs et ouverture des dispositifs
3. Tri topologique
4. Création de la machine réactive

La première phase consiste à aplatir l'arbre des configurations afin de placer tous les dispositifs au même niveau. L'algorithme, appelé décomposition totale, a déjà été décrit en annexe A (algorithme A.1). La seconde phase consiste à créer les structures de base qui serviront à l'exécution : les Valeurs (signaux valués) et les Processeurs. Un tri topologique est ensuite effectué sur les processeurs, à partir du graphe de dépendances entre dispositifs (→ section A.3.5 page 204). La dernière phase consiste à construire la machine réactive et remplir sa structure.

Nous ne décrirons pas le mécanisme d'aplatissement (déjà décrit en Annexe A) et le tri topologique (déjà connu [LACOMME *et al.* 03]). En revanche nous détaillerons la phase consistant à créer les «

structures de base », après avoir décrit ces structures. Puis nous décrirons les machines réactives et la façon dont elles sont créées.

C.3.1 Codage des signaux d'entrée et des processeurs

Dans la section précédente, nous avons introduit et décrit les processeurs à l'aide d'outils mathématiques simples. Dans cette partie, nous emploierons des représentations alternatives sous forme de structures de données.

Un signal valué est codé par une structure nommée `Valeur`, qui encapsule une *valeur de variable* et un *temps*. Le temps est un entier précisant à quel moment le signal valué a reçu un signal pour la dernière fois. Cette représentation est équivalente à celle des signaux valués vus précédemment : un signal est présent ($s = vrai$) si et seulement si le temps de la Valeur est égal au temps courant de la machine réactive (i.e. au temps global).

La structure `Valeur` est décrite ci-dessous :

Valeur	
`parent`	**Machine**
`valIndex`	entier
`valeur`	variable
`temps`	entier
`signal`	vrai/faux

TAB. C.1 – Structure d'un signal valué.

- `parent :` La machine réactive à laquelle appartient cette Valeur.
- `valIndex :` Spécifie l'indice de cette Valeur dans la machine réactive.
- `valeur :` La *valeur de variable* du signal valué, stockée dans une variable au sens langage de programmation. La représentation concrète de cette variable dépend du type du slot associé, ainsi que du langage de programmation choisi pour l'implémentation. Nous ne nous en préoccupons pas ici.
- `temps :` Le *temps* associé à la Valeur, spécifiant à quel moment elle a reçu un signal pour la dernière fois.
- ► `signal :` Attribut booléen dérivé spécifiant si un signal est présent. Cet attribut vaut `vrai` ssi le temps de la valeur est égal au temps de la machine réactive parente.

Un processeur est codé par une structure `Processeur`. Cette structure comporte des références à des Valeurs d'entrée et de sortie, ainsi qu'une décomposition de $P_{E,S}$ en deux séries de fonctions nommées *calculValeur* et *calculSignal*. Ces fonctions sont des projections de la fonction $P_{E,S}$ sur les valeurs et les signaux de son ensemble d'arrivée :

Soit $P_{E,S} : h^t(E) \mapsto v(S) = \langle \langle x_1, s_1 \rangle, \ldots, \langle x_n, s_n \rangle \rangle$.
$calculValeur[i] : h^t(E) \mapsto x_i$
$calculSignal[i] : h^t(E) \mapsto s_i$

- `entrées :` Cette table contient les Valeurs représentant les signaux d'entrée du processeur.
- `sorties :` Cette table contient les Valeurs représentant les signaux de sortie du processeur.

Processeur	
`entrées`	**Valeur** [ ]
`sorties`	**Valeur** [ ]
calculValeur	*fonction* [ ]
calculSignal	*fonction* [ ]

TAB. C.2 – Structure d'un processeur.

- *calculValeur* **:** Les projections de la fonction *P* sur les valeurs de variable de chaque signal de sortie. Cette table comporte autant de fonctions que d'éléments dans la table `valeursSortie`.
- *calculSignal* **:** Les projections de la fonction *P* sur les signaux de chaque signal de sortie. Cette table comporte autant de fonctions que d'éléments dans la table `valeursSortie`.

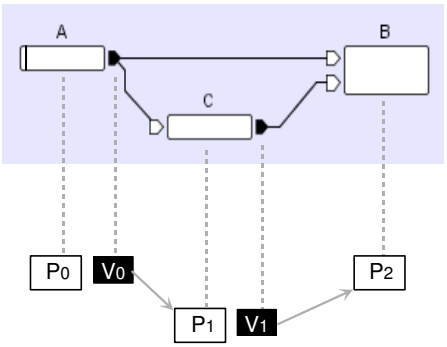
C.3.2 Création des valeurs et ouverture des dispositifs

La seconde étape phase du lancement d'une configuration d'entrée consiste à créer les structures contenant les valeurs des slots.

Une structure `Valeur` est créée *pour tout slot de sortie* de la configuration d'entrée. Les slots d'entrée ne comportent pas de Valeur (voir la figure C.3 page 227), mais référencent celle de leur slot connecté. Il n'y aura donc pas à proprement parler de propagation de valeur entre deux slots, puisque ceux-ci partageront la même valeur.

Une fois les valeurs créées, les dispositifs sont *ouverts* : lorsqu'un dispositif est ouvert, celui-ci retourne une structure Processeur créée à partir de sa *fonction d'exécution*.

C.3.3 Construction de la machine réactive



ProcInfo	**0**	**1**	**2**
Processeur	P_0	P_1	P_2
actif	V	F	F
déclenché	V	F	F

ValInfo	**0**	**1**
Valeur	V_0	V_1
variable	-	-
temps	0	0
depFirst	0	2
depLast	1	2

DepInfo	**0**	**1**	**2**
procIndex	2	1	2

temps

FIG. C.4 – Exemple de machine réactive créée à partir d'une configuration exemple.

La machine réactive gère tous les aspects de l'exécution des configurations d'entrée, et en particulier le *déclenchement* des processeurs et la propagation des valeurs. Un exemple de structure de machine réactive est illustré sur la figure C.4 page précédente, qui reprend la configuration-exemple de la figure C.3 page 227. Cette structure est la suivante :

Machine	
`ProcInfo`	(**Processeur**, vrai/faux, vrai/faux) [ ]
`ValInfo`	(**Valeur**, entier, entier) [ ]
`DepInfo`	vrai/faux [ ]
`temps`	entier

TAB. C.3 – Structure d'une machine réactive.

- `ProcInfo :` Cette table contient une liste topologiquement ordonnée des processeurs avec leurs conditions de déclenchement. Le booléen `actif` spécifie si le processeur doit être continuellement déclenché. La valeur de cet attribut dépend de la nature de la fonction $P_{E,S}$ du processeur (→ section C.2.4 page 226). Le second booléen, `déclenché` est interne à l'algorithme d'exécution. Initialement égal à `actif`, il servira à spécifier pour l'itération courante si les valeurs du processeur doivent être mises à jour.
- `ValInfo :` Cette table contient la liste des Valeurs ainsi que leurs dépendances (illustrées sur la figure C.4 page précédente par des flèches pleines). Les dépendances d'une Valeur sont définies par les attributs `depFirst` et `depLast`, qui référencent par leur index un ensemble d'éléments contigus dans la table `DepInfo`.
- `DepInfo :` Cette table contient la liste des dépendances référencées par `OutInfo`, et est remplie en même temps qu'elle. Lorsqu'un slot de sortie est traité par l'ajout d'une Valeur dans `OutInfo`, une dépendance est créé pour chaque slot d'entrée qui lui est directement connecté. Une dépendance est uniquement décrite par l'attribut `ProcIndex`, qui référence le processeur du dispositif parent du slot d'entrée, en pointant vers son index dans la table `ProcInfo`.
- `temps :` Cet attribut est interne à l'algorithme d'exécution au même titre que l'attribut `déclenché` de la première table. Initialement à zéro, ce compteur entier spécifiera le temps courant de la machine réactive.

C.4 Algorithme d'exécution

L'exécution est partagée entre les Processeurs qui mettent à jour leurs Valeurs de sortie, les Valeurs qui notifient la machine réactive, et enfin la machine réactive qui synchronise l'ensemble.

C.4.1 Mise à jour d'un processeur

La mise à jour d'un Processeur consiste à modifier ses valeurs de sortie, et à notifier la machine réactive parente afin que cette mise à jour soit propagée. L'ensemble de ce mécanisme est décrit par l'algorithme C.1 page suivante : la procédure *miseAJour* définie dans la structure `Processeur` applique les fonctions *calculValeur* et *calculSignal* à ses Valeurs de sortie. Elle délègue les assignations finales et les notifications à la procédure *miseAJour* définie dans la structure `Valeur`. Lorsqu'un signal est présent, cette dernière appelle la procédure *envoiSignal* de la machine réactive, décrite dans la section suivante.

```
proc Processeur.miseAJour()
   for i=0 to sorties.length do
      sorties[i].miseAJour(calculValeur[i](entrées), calculSignal[i](entrées))
   end for
end proc

proc Valeur.miseAJour(nouvelleValeur, signal)
   valeur ← nouvelleValeur
   if signal = vrai then
      machine.envoiSignal(valIndex)
   end if
end proc
```

Algorithme C.1: Mise à jour d'un processeur.

C.4.2 La boucle d'exécution

L'exécution d'une machine réactive consiste en une série d'itérations appelées *ticks*. Pendant un tick, tous les signaux sont propagés dans la machine réactive. La procédure *tick* (algorithme C.2) décrit une itération : la structure `ProcInfo` est parcourue dans l'ordre, et chaque processeur est mis à jour s'il est *déclenché*. La mise à jour d'un dispositif peut amener le déclenchement d'autres processeurs situés après lui dans la table.

```
proc Machine.tick()
   temps++
   for i=0 to ProcInfo.length do
      pinfo ← ProcInfo[i]
      if pinfo.déclenché then
         pinfo.Processeur.miseAJour()
      end if
      if not pinfo.actif then
         pinfo.déclenché ← faux
      end if
   end for
end proc

proc Machine.envoiSignal(int valIndex)
   dinfo ← DepInfo[valIndex]
   dinfo.Valeur.temps ← temps
   for i=dinfo.depFirst to dinfo.depLast do
      ProcInfo[DepInfo[i].procIndex].déclenché ← vrai
   end for
end proc
```

Algorithme C.2: Itération d'une machine réactive.

Cet algorithme de propagation assure qu'à la fin d'un tick, tous les changements ont correctement été propagés, et que la mise à jour d'un dispositif n'est *jamais effectuée plus d'une fois* : dans la configuration-exemple de la figure C.4 page 229, quand *B* reçoit une valeur du dispositif actif *A*, il attendra la mise à jour de *C* avant de se mettre à jour.

L'exécution d'une configuration d'entrée consiste en des appels successifs de la procédure *tick*. Notre modèle ne décrit pas comment ces appels sont effectués, en particulier s'ils une pause les sépare (pour laisser la main à d'autres processus ou garder une fréquence constante), ou encore s'ils sont appelés uniquement lorsque des valeurs sont présentes en entrée (interruptions matérielles).

C.5 L'environnement

Le modèle d'exécution que nous avons décrit est particulièrement bien adapté à la description de traitements de données *déterministes*. Cependant, un système interactif comporte souvent une part de non-déterminisme, en particulier du non-déterminisme temporel résultant de processus asynchrones, qui « prennent du temps ». En outre, il est évident que notre système nécessite d'être alimenté en informations, en particulier celles provenant des dispositifs d'entrée, et doit également « agir » à l'extérieur du système, soit sur l'application contrôlée, soit pour générer un retour utilisateur (feedback). Tous ces mécanismes ne peuvent être pris en compte sans inclure dans le modèle la notion d'*environnement*, système externe à la machine réactive.

Nous évoquerons dans cette partie comment nous pouvons, dans notre modèle et dans une éventuelle implémentation, prendre en compte cette communication, puis nous aborderons un autre problème lié à l'environnement, celui de l'hypothèse réactive.

C.5.1 Communication avec l'environnement : non-déterminisme et effets de bord

Lorsqu'un dispositif ne possède pas d'entrées ou de sorties *implicites*, son comportement lors de l'exécution est entièrement déterminé par un processeur opérant uniquement sur les signaux d'entrée et de sortie liés aux slots du dispositif. Dans le cas contraire, le dispositif communique avec l'*environnement*, et ce processeur ne peut décrire intégralement son comportement.

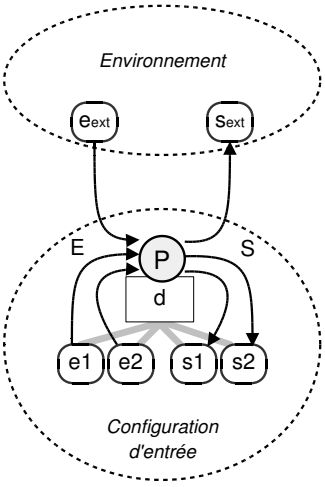


FIG. C.5 – Communication entre un processeur et un système extérieur à la configuration, appelé environnement.

Nous pouvons cependant généraliser notre modèle déterministe en supposant que l'environnement

est accessible à travers deux signaux valués (dont les types peuvent bien sûr être complexes) : le signal valué e_{ext} et le signal valué s_{ext}. Le dispositif interagit alors avec l'environnement en lisant e_{ext} (s'il comporte des entrées implicites) et en écrivant dans s_{ext} (s'il comporte des sorties implicites). Le comportement d'un tel dispositif peut alors être décrit avec un processeur $P_{E,S}$ défini sur les signaux valués relatifs aux slots auxquels on ajoute les signaux valués de l'environnement (figure C.5 page ci-contre) :

$$E = E_{slots} \cup \{e_{ext}\}$$
$$S = S_{slots} \cup \{s_{ext}\}$$

Notre modèle d'exécution décrit le comportement des configurations d'entrée, mais non le comportement de l'environnement (qui, par définition, ne fait pas partie de notre système), ni les mécanismes de communication avec cet environnement. En particulier, les variables associées à e_{ext} et s_{ext} ne sont pas explicites et ni leur type ni leur valeur ne sont connues : elles permettent uniquement d'évoquer, dans notre modèle, la présence de non-déterminisme (pour le premier) et d'effets de bord (pour le second).

Dans une implémentation concrète, la communication avec l'environnement (librairies et parties du code externes à la machine réactive) sera implémentée librement dans la fonction de mise à jour, mais devront cependant être explicitement déclarés par la présence d'entrées et de sorties implicites. La présence d'entrées implicites implique souvent un dispositif *actif*, qui a besoin de lire l'environnement même en l'absence de signaux en entrée : il y a donc un sens à ce qu'un dispositif à entrées implicites soit actif par défaut. La présence de sorties implicites n'a quant à lui pas d'influence sur les mécanismes d'exécution, mais peut apporter (tout comme les entrées implicites) une information utile sur la sémantique du dispositif dans une éventuelle représentation graphique.

C.5.2 Ouverture non-déterministe des dispositifs

Nous avons décrit la phase d'ouverture des dispositifs comme un mécanisme permettant de créer des processeurs de manière déterministe. Cependant, les dispositifs comportant des entrées ou des sorties implicites ont souvent besoin d'allouer au sein de l'environnement des ressources pour préparer leur exécution. Ces mécanismes d'initialisation peuvent, de manière non-déterministe, *réussir* ou *échouer* (dans le cas des dispositifs sans entrées ou sorties implicites, nous supposerons qu'elle réussit toujours).

Lorsque l'ouverture d'un dispositif d a réussi, celui-ci retourne, comme décrit précédemment, un Processeur construit à partir de sa fonction d'exécution $P = \varepsilon(\Pi(d))$. Dans le cas contraire, il retourne le *Processeur nul* décrit par la fonction $P_\varnothing$:

$$P_\varnothing : h^t(E) \mapsto \varnothing$$

C.5.3 L'hypothèse réactive dans ICoM

Dans une implémentation concrète, il est évident que le traitement de l'information (effectué par l'ensemble des processeurs) prend un certain temps, ainsi que la communication entre ces processeurs (pris en charge par la machine réactive). En pratique, cette dernière est souvent négligeable par rapport aux traitements de données. La durée d'un tick dans une machine réactive est par conséquent au moins égale à la somme des durées des traitements effectués par les processeurs. En outre, le temps

de réponse d'une machine réactive augmente linéairement avec la taille d'une configuration d'entrée.

Un modèle réactif ne reste valide que tant que l'hypothèse du synchronisme parfait est vérifiée, c'est-à-dire si la machine réactive réagit *plus vite que l'environnement*. Dans notre modèle, le signal valué e_{ext} est synchronisé sur l'horloge de la machine réactive. En supposant que e_{ext} « lit » des signaux que l'environnement lui envoie selon son horloge propre, l'hypothèse réactive est vérifiée si durant chaque tick, au plus un signal est émis par l'environnement.

e_{ext} décrit en général l'information provenant d'un dispositif d'entrée concret. En supposant, pour simplifier, que ce dispositif concret émet des informations à fréquence constante, la machine réactive doit réagir au moins à la même fréquence[1]. Pour une configuration d'entrée comportant plusieurs de ces dispositifs, la fréquence de réaction de la machine réactive doit être au moins égale à celle du dispositif émettant à la plus haute fréquence.

[1] Il n'est cependant pas nécessaire qu'une machine réactive réagisse à fréquence constante.

Bibliographie

[ACCOT *et al.* 98] Johnny ACCOT, Stéphane CHATTY, Yannick JESTIN, et Stéphane SIRE. « Conception des interfaces : et si nous analysions enfin la tâche du programmeur ? ». Dans *Prise de position pour les dixièmes journées francophones sur l'Interaction Homme Machine (IHM'98)*, septembre 2–4 1998.

[ACCOT *et al.* 97] Johnny ACCOT, Stéphane CHATTY, Sébastien MAURY, et Philippe PALANQUE. « Formal transducers : Models of devices and building bricks for the design of highly interactive systems ». Dans M. D. HARRISON et J. C. TORRES, éditeurs, *Design, Specification and Verification of Interactive Systems '97*, Eurographics, pages 143–159, Wien, 1997. Springer-Verlag. Proceedings of the Eurographics Workshop in Granada, Spain, June 4 – 6, 1997.

[ALIAS 01] « Maya real-time author (RTA) ». Alias Systems (Alias|Wavefront), décembre 2001. `http://www.alias.com`.

[APPERT *et al.* 03] Caroline APPERT, Michel BEAUDOUIN-LAFON, et Wendy MACKAY. « Context matters : Evaluating interaction techniques with the CIS model ». Rapport Technique 1372, Laboratoire de Recherche en Informatique (LRI), Université de Paris-Sud, 2003.

[APPLE 02] « Introduction to the aqua human interface ». Apple Computer, Inc., 2002. `http://developer.apple.com`.

[APPLE 03] « Interface builder ». Apple Computer, Inc., 2003. `http://developer.apple.com/tools/interfacebuilder/`.

[BALAKRISHNAN *et al.* 99] Ravin BALAKRISHNAN, George FITZMAURICE, Gordon KURTENBACH, et William BUXTON. « Digital tape drawing ». Dans *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology*, pages 161–170, N.Y., novembre 7–10 1999. ACM Press.

[BARLOW *et al.* 90] Horace BARLOW, Colin BLAKEMORE, et Miranda WESTON-SMITH. *Images and Understanding : Thoughts about images : Ideas about understanding*. Cambridge University Press, 1990. ISBN 0-521-36944-4.

[BASTIDE *et al.* 02] Rémi BASTIDE, David NAVARRE, et Philippe PALANQUE. « A model-based tool for interactive prototyping of highly interactive applications ». Dans *CHI '02 extended abstracts on Human factors in computer systems*, pages 516–517. ACM Press, 2002.

[BAUDEL 95] Thomas BAUDEL. *Morphologie de l'Interaction Humain-Ordinateur : Étude de Modèles d'Interaction Gestuelle*. Thèse de doctorat, Université de Paris-Sud, U.F.R. scientifique d'Orsay, dec 1995.

[BAUDISCH *et al.* 03] P. BAUDISCH, E. CUTRELL, D. ROBBINS, M. CZERWINSKI, P. TANDLER, B. BEDERSON, et A. ZIERLINGER. « Drag-and-pop and drag-and-pick : Techniques for accessing remote screen content on touch- and pen-operated systems ». Dans *Proceedings of the 9th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'03)*, pages 57–64, septembre 2003.

[BEAUDOUIN-LAFON 97] Michel BEAUDOUIN-LAFON. « Interaction instrumentale : de la manipulation directe à la réalité augmentée ». Dans *Actes des Neuvièmes Journées sur l'Interaction Homme-Machine, IHM'97*, sep 1997.

[BEAUDOUIN-LAFON 00] Michel BEAUDOUIN-LAFON. « Instrumental interaction : an interaction model for designing post-WIMP user interfaces ». Dans Thea TURNER, Gerd SZWILLUS, Mary CZERWINSKI, et Paternò FABIO, éditeurs, *Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI-00)*, pages 446–453, N. Y., avril 1–6 2000. ACM Press.

[BEAUDOUIN-LAFON *et al.* 90] Michel BEAUDOUIN-LAFON, Yves BERTEAUD, et Stéphane CHATTY. « Creating direct manipulation applications with xtv ». Dans *Proc. European X Window System Conference*, novembre 1990.

[BEAUDOUIN-LAFON & LASSEN 00] Michel BEAUDOUIN-LAFON et Henry Michael LASSEN. « The architecture and implementation of CPN2000, a post-WIMP graphical application ». Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 181–190, N.Y., novembre 5–8 2000. ACM Press.

[BEDERSON 03] Ben BEDERSON. « The piccolo 1.0 toolkit ». Human Computer Interaction Lab (HCIL), University of Maryland, avril 2003. `http://www.cs.umd.edu/hcil/piccolo/`.

[BEDERSON *et al.* 00] Benjamin B. BEDERSON, Jon MEYER, et Lance GOOD. « Jazz : an extensible zoomable user interface graphics toolkit in java ». Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 171–180, N.Y., novembre 5–8 2000. ACM Press.

[BERRY 89] Gérard BERRY. « Real time programming : special purpose or general purpose languages ». Rapport interne RR-1065, Inria, Institut National de Recherche en Informatique et en Automatique, 1989.

[BERRY 99] Gérard BERRY. « The Esterel v5 language primer ». Rapport Technique, avril 1999. `http://www-sop.inria.fr/meije/esterel/doc/main-papers.html`.

[BERRY 00] Gérard BERRY. *The Foundations of Esterel*. MIT Press, 2000.

[BERRY *et al.* 87] Gérard BERRY, P. COURONNÉ, et G. GONTHIER. « Programmation synchrone des systèmes réactifs : le langage ESTEREL ». *Technique et Science Informatique*, 6(4), 1987.

[BIER *et al.* 93] E. BIER, M. STONE, K. PIER, W. BUXTON, et T. DEROSE. « Toolglass and magic lenses : The see-through interface ». *Proceedings of SIGGRAPH'93*, pages 73–80, 1993.

[BIER & FREEMAN 91] E. A. BIER et S. FREEMAN. « MMM : A user interface architecture for shared editors on a single screen ». Dans *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology (UIST '91)*, pages 79–86, South Carolina, USA, novembre 11-13 1991. ACM Press.

[BIER & STONE 86] E. A. BIER et M. C. STONE. « Snap-dragging ». *Computer Graphics (Proc. ACM SIGGRAPH '86)*, 20(4) :233–240, 1986.

[BLANCH 02] Renaud BLANCH. « Programmer l'interaction avec des machines états hiérarchiques ». Dans *Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine)*, pages 129–136. ACM Press, 2002.

[BOLT 80] Richard A. BOLT. « "put-that-there" : Voice and gesture at the graphics interface ». *Computer Graphics*, 14(3) :262–270, juillet 1980.

[BORNING 79] Alan Hamilton BORNING. *Thinglab - A Constraint-Oriented Simulation Laboratory*. PhD thesis, Stanford University, juillet 1979. Également disponible comme rapports de recherche STAN-CS-79-746 (Stanford Computer Science Department) et SSL-79-3 (XEROX Palo Alto Research Center).

[BOURIT 00] Cyril BOURIT. « Chamois : un logiciel de constructions géométriques ». août 2000. http://membres.lycos.fr/bourit/.

[BUXTON 83] William BUXTON. « Lexical and pragmatic considerations of input structures ». *Computer Graphics*, 17(1) :31–37, janvier 1983.

[BUXTON 86A] William BUXTON. « Chunking and phrasing and the design of human-computer dialogues ». Dans H. J. KUGLER, éditeur, *Information Processing '86, Proceedings of the IFIP 10th Worm Computer Congress*, pages 475–480. North Holland Publishers, 1986.

[BUXTON 86B] William BUXTON. « There is more to interaction than meets the eye : Some issues in manual input ». Dans D. A. NORMAN et S. W. DRAPER, éditeurs, *User Centred System Design : New Perspectives on Human-computer Interaction*, pages 319–337. Lawrence Erlbaum Associates, Hillsdale, NJ., 1986.

[BUXTON & MYERS 86] William BUXTON et Brad A. MYERS. « A study in two-handed input ». *Human Factors in Computing Systems*, pages 321–326, 1986.

[CADOZ 94] Claude CADOZ. « Le geste, canal de communication homme/machine : la communication instrumentale ». *Technique et Science de l'Information*, 13(1) :31–61, 1994.

[CAPPONI & LABORDE 91] B. CAPPONI et C. LABORDE. « Cabri-géomètre, un environnement pour l'apprentissage de la géomètrie élémentaire ». Dans *Actes de la VIème école d'été de didactique des mathématiques*, pages 220–22, 1991.

[CARD *et al.* 90] Stuart K. CARD, Jock D. MACKINLAY, et George G. ROBERTSON. « The design space of input devices ». Dans *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, Multi-Media, pages 117–124, 1990.

[CARD *et al.* 91] Stuart K. CARD, Jock D. MACKINLAY, et George G. ROBERTSON. « A morphological analysis of the design space of input devices ». *ACM Trans. on Inf. Sys.*, 9(2) :99, 1991.

[CARR 91] R. M. CARR. « The point of the pen (PenPoint operating system) ». *Byte Magazine*, 16(2) :211–214, 216, 219–221, février 1991.

[CARSON 97] George S. CARSON. « Standards pipeline :the Open GL specification ». *Computer Graphics*, 31(2) :17–18, mai 1997.

[CHAPMAN 03] Davis CHAPMAN. *Visual C++ 6*. Le Programmeur. CampusPress, mars 2003. ISBN 2744015571.

[CHATTY 94] Stephane CHATTY. « Extending a graphical toolkit for two-handed interaction ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Two Hands and Three Dimensions, pages 195–204, 1994.

[CMI 02] « Le projet GINA (Géométrie Interactive et NAturelle) ». École des Mines de Nantes, équipe CMI, 2002. http://www.emn.fr/x-info/gina/.

[COHEN *et al.* 89] P. R. COHEN, M. DALRYMPLE, D. B. MORAN, F. C. PEREIRA, et J. W. SULLIVAN. « Synergistic use of direct manipulation and natural language ». Dans *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 227–233. ACM Press, 1989.

[COUTAZ 87] Joelle COUTAZ. « PAC, an object-oriented model for dialog design ». Dans *Proceedings of IFIP INTERACT'87 : Human-Computer Interaction*, 2. Design and Evaluation Methods : 2.5 Dialogue Design and Evaluation, pages 431–436, 1987.

[CYCORE 03] « Cult3D Designer ». Cycore, 2003. http://www.cult3d.com/Cult3D/.

[DIX & RUNCIMAN 85] Alan DIX et Colin RUNCIMAN. « Abstract models of interactive systems ». Dans *Proceedings of the HCI'85 Conference on People and Computers : Designing the Interface*, The Design Process : Models and Notation for Interaction, pages 13–22, 1985.

[DOHERTY *et al.* 01] Eamon DOHERTY, Gilbert COCKTON, Chris BLOOR, et Dennis BENIGNO. « Improving the performance of the cyberlink mental interface with "yes / no program" ». Dans *Proceedings of ACM CHI 2001 Conference on Human Factors in Computing Systems*, Motion and Emotion, pages 69–76, 2001.

[DRAGICEVIC 98] Pierre DRAGICEVIC. « Concrétiser les dispositifs d'entrée dans les outils de développement », Prise de position. Dans *Actes des dixièmes journées francophones sur l'Interaction Homme Machine (IHM'98)*, pages 133–139, septembre 2–4 1998.

[DRAGICEVIC 01] Pierre DRAGICEVIC. « Une architecture en cascade pour des systèmes interactifs multi-dispositifs », Rencontres Doctorales, papier court. Dans A. BLANDFORD, J. VANDERDONKT, et P. GRAY, éditeurs, *People and Computers XV Interaction without Frontiers : Joint proceedings of IHM 2001 and HCI 2001*, volume 2, pages 191–192. Springer Verlag, 2001.

[DRAGICEVIC 02] Pierre DRAGICEVIC. « Page Web du projet ICON ». École des Mines de Nantes, équipe CMI, 2002. `http://www.emn.fr/x-info/icon/`.

[DRAGICEVIC & FEKETE 99] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « Étude d'une boîte à outils multi-dispositifs ». Dans *Actes de la 11ième conférence francophone d'Interaction Homme-Machine (IHM99)*, pages 55–62. Cepadues, novembre 1999.

[DRAGICEVIC & FEKETE 00] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « Input device selection and interaction configuration with ICON ». Rapport Technique 00/5/INFO, École des Mines de Nantes, département Informatique, novembre 2000.

[DRAGICEVIC & FEKETE 01] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « Input device selection and interaction configuration with ICON ». Dans A. BLANDFORD, J. VANDERDONKT, et P. GRAY, éditeurs, *People and Computers XV Interaction without Frontiers : Joint proceedings of IHM 2001 and HCI 2001*, pages 543–558. Springer Verlag, 2001.

[DRAGICEVIC & FEKETE 02] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « ICON : Input device selection and interaction configuration », Démonstration, papier court. Dans *UIST '02 Companion. The 15th Annual ACM Symposium on User Interface Software and Technology*, octobre 27–30 2002.

[DRAGICEVIC & FEKETE 03] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « ICON : Towards high input adaptability of interactive applications ». Rapport Technique (en cours de publication), École des Mines de Nantes, département Informatique, décembre 2003.

[DRAGICEVIC & FEKETE 04] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « The input configurator toolkit : Towards high input adaptability in interactive applications ». Dans SOUMIS À *AVI'2004 Advanced Visual Interfaces*. ACM Press, mai 25–28 2004.

[DRAGICEVIC & HUOT 02] Pierre DRAGICEVIC et Stéphane HUOT. « SpiraClock : a continuous and non-intrusive display for upcoming events ». Dans *CHI '02 extended abstracts on Human factors in computer systems*, pages 604–605. ACM Press, 2002.

[DUKE & HARRISON 93] D. J. DUKE et M. D. HARRISON. « Abstract interaction objects ». Dans R. J. HUBBOLD et R. JUAN, éditeurs, *Eurographics '93*, pages 25–36, Oxford, UK, 1993. Eurographics, Blackwell Publishers.

[DUNN 00] Jeff DUNN. « Developing accessible JFC applications ». Sun Microsystems' Accessibility Team, juin 2000. `http://www.sun.com/access/developers/developing-accessible-apps/`.

[DUNN & HERZOG 77] Robert M. DUNN et Bertram HERZOG. « Status report of the Graphics Standards Planning Committee of ACM/SIGGRAPH ». *Computer Graphics*, 11(3) :I–19 + II–117, Fall 1977.

[ECKERT *et al.* 79] R. ECKERT, G. ENDERLE, K. KANSY, et F.J. PRESTER. « GKS'79 - proposal of a standard for a graphical kernel system ». Dans *Proceedings of Eurographics'79*, 1979.

[ECKSTEIN & LOY 02] Robert ECKSTEIN et Marc LOY. *Java Swing*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 2e édition, 2002. ISBN 0-596-00408-7.

[ELLIOTT *et al.* 94] Conal ELLIOTT, Greg SCHECHTER, Ricky YEUNG, et Salim ABI-EZZI. « TBAG : A high level framework for interactive, animated 3D graphics applications ». Dans Andrew GLASSNER, éditeur, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 421–434. ACM SIGGRAPH, ACM Press, juillet 1994. ISBN 0-89791-667-0.

[ELO 02] « Keys to a successful kiosk application ». Elo TouchSystems, Inc., 2002. `http://www.elotouch.com`.

[ESTEBAN 97] Olivier ESTEBAN. *Programmation visuelle pour la construction d'interfaces homme-machine hautement interactives*. Thèse de doctorat, Laboratoire Interface Homme Systèmes (LIHS), avril 1997.

[ESTEBAN *et al.* 95] O. ESTEBAN, S. CHATTY, et P. PALANQUE. « Whizz'ed : a visual environment for building highly interactive software ». Dans *Proceedings of IFIP INTERACT'95 : Human-Computer Interaction*, pages 121–126, 1995.

[FEKETE & BEAUDOUIN-LAFON 96] Jean-Daniel FEKETE et Michel BEAUDOUIN-LAFON. « Using the multi-layer model for building interactive graphical applications ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Papers : Tools, pages 109–118, 1996.

[FEKETE & DRAGICEVIC 00] Jean-Daniel FEKETE et Pierre DRAGICEVIC. « Une architecture multi-dispositifs ». Dans *Actes du Colloque sur les Interfaces Multimodales*, mai 9–10 2000.

[FEKETE *et al.* 98] Jean-Daniel FEKETE, Martin RICHARD, et Pierre DRAGICEVIC. « Specification and verification of interactors : A tour of esterel ». Dans Chris R. ROAST, éditeur, *Formal Aspects of Human Computer Interaction FAHCI'98*, pages 103–118. British Computer Society, septembre 1998.

[FEKETE 96A] Jean-Daniel FEKETE. « Les trois services du noyau sémantique indispensables à l'IHM ». Dans *Actes Huitièmes Journées sur l'Ingénierie des Interfaces Homme-Machine, IHM'96*, pages 45–50, septembre 1996.

[FEKETE 96B] Jean-Daniel FEKETE. *Un modèle multicouche pour la construction d'applications graphiques interactives*. Thèse de doctorat, Université de Paris-Sud, Orsay (France), janvier 1996.

[FIGUEROA *et al.* 02] Pablo FIGUEROA, Mark GREEN, et H. James HOOVER. « InTml : a description language for VR applications ». Dans *Proceeding of the seventh international conference on 3D Web technology*, pages 53–58. ACM Press, 2002.

[FINGERWORKS 03] « iGesture multitouch Pad ». FingerWorks, 2003. Brevet US. 6323846. `http://www.fingerworks.com/`.

[FITZMAURICE & BUXTON 97] George FITZMAURICE et William BUXTON. « An empirical evaluation of graspable user interfaces : Towards specialized, space-multiplexed input ». Dans *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 de *PAPERS : Handy User Interfaces*, pages 43–50, 1997.

[FITZMAURICE *et al.* 95] George W. FITZMAURICE, Hiroshi ISHII, et William BUXTON. « Bricks : Laying the foundations for graspable user interfaces ». Dans Irvin R. KATZ, Robert MACK, Linn MARKS, Mary Beth ROSSON, et Jakob NIELSEN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI'95)*, pages 442–449, New York, NY, USA, mai 1995. ACM Press.

[FOLEY *et al.* 90] J. FOLEY, A. VAN DAM, S. FEINER, et J. HUGHES. *Computer graphics Principles and Practice*. Addison-Wesley, 2e édition, 1990.

[FOLEY *et al.* 84] James D. FOLEY, Victor L. WALLACE, et Peggy CHAN. « The human factors of computer graphics interaction techniques ». *IEEE Computer Graphics and Applications*, 4(11) :13–48, novembre 1984.

[FOWLER 99] A. FOWLER. « A Swing architecture overview : The inside story on JFC component design ». Sun Microsystems, 1999. `http://java.sun.com/products/jfc/tsc/articles/architecture/`.

[FRANTZ 00] Gérard FRANTZ. *Visual Basic 6 : Le guide du programmeur*. La référence. Osman Eyrolles Multimédia, novembre 2000. ISBN 2746402343.

[GILL 02] Lisa GILL. « The secret of logitech's success ». oct 2002. E-Commerce Times. `http://www.ecommercetimes.com/perl/story/19664.html`.

[GOLDBERG & ROBSON 81] A. GOLDBERG et D. ROBSON. « The Smalltalk-80 system ». *Byte Magazine*, 6(8) :36–48, août 1981.

[GREENBERG & BOYLE 02] Saul GREENBERG et Michael BOYLE. « Customizable physical interfaces for interacting with conventional applications ». Dans *Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST-02)*, pages 31–40, New York, octobre 27–30 2002. ACM Press.

[GREENBERG & FITCHETT 01] Saul GREENBERG et Chester FITCHETT. « Phidgets : Easy development of physical interfaces through physical widgets ». Dans *Proceedings of the 14th Annual Symposium on User Interface Software and Technology (UIST-01)*, pages 209–218, New York, novembre 11–14 2001. ACM Press.

[GSA 91] GSA. *Managing end user computing for users with disabilities*. General Services Administration, 1991. Washington, DC.

[GUIARD 87] Yves GUIARD. « Asymmetric division of labor in human skilled bimanual action : The kinematic chain as a model. ». *The Journal of Motor Behavior*, 19(4) :486–517, 1987.

[HALBWACHS *et al.* 91] N. HALBWACHS, P. CASPI, P. RAYMOND, et D. PILAUD. « The synchronous data flow programming language LUSTRE ». Dans *Proceedings of the IEEE*, volume 79, septembre 1991.

[HAPTEK 03] « PeoplePutty : dynamic 3-D character building tool ». Haptek, Inc., 2003. `http://www.haptek.com/`.

[HAREL 87] David HAREL. « Statecharts : A visual formalism for complex systems ». *Science of Computer Programming*, 8(3) :231–274, juin 1987.

[HEWITT 84] W. T. HEWITT. « PHIGS — Programmer's Hierarchical Interactive Graphics System ». *Computer Graphics Forum*, 3(4) :299–300, décembre 1984.

[HINCKLEY *et al.* 98] Ken HINCKLEY, Mary CZERWINSKI, et Mike SINCLAIR. « Interaction and modeling techniques for desktop two-handed input ». Dans *Proceedings of the 11th Annual Symposium on User Interface Software and Technologie (UIST-98)*, pages 49–58, New York, novembre 1–4 1998. ACM Press.

[HINCKLEY *et al.* 03] Ken HINCKLEY, R.J.K. JACOB, et C. WARE. *Computer Science and Engineering Handbook Second Edition (in press)*, Chapitre Input/Output Devices and Interaction Techniques. CRC Press, 2e édition, 2003.

[HINCKLEY *et al.* 94A] Ken HINCKLEY, R. PAUSCH, J. C. GOBLE, et N. F. KASSELL. « Passive real-world interface props for neurosurgical visualization ». *Proceedings of CHI '94*, pages 452–458, 1994.

[HINCKLEY *et al.* 94B] Ken HINCKLEY, Randy PAUSCH, John C. GOBLE, et Neal F. KASSELL. « A survey of design issues in spatial input ». Dans *ACM UIST'94 Symp. on User Interface Software & Technology*, pages 213–222. 1994.

[HONEYWELL 99] Steve HONEYWELL. *Quake III Arena : Prima's Official Strategy Guide*. Prima Lifestyles, dec 1999. ISBN 0761525882.

[HONG & LANDAY 00] Jason I. HONG et James A. LANDAY. « SATIN : a toolkit for informal ink-based applications ». Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 63–72, N.Y., novembre 5–8 2000. ACM Press.

[HOURCADE & BEDERSON 99] Juan Pablo HOURCADE et Benjamin B. BEDERSON. « Architecture and implementation of a java package for multiple input devices (MID) ». Rapport interne CS-TR-4018, University of Maryland, College Park, mai 1999.

[HOURIZI & JOHNSON 01] R. HOURIZI et P. JOHNSON. « Beyond mode error : Supporting strategic knowledge structures to enhance cockpit safety ». Dans A. BLANDFORD, J. VANDERDONKT, et P. GRAY, éditeurs, *People and Computers XV Interaction without Frontiers : Joint proceedings of IHM 2001 and HCI 2001*, 2001.

[HUDSON 90] Scott E. HUDSON. « Adaptive semantic snapping - a technique for semantic feedback at the lexical level ». Dans *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 65–70. ACM Press, 1990.

[HUDSON *et al.* 97] Scott E. HUDSON, Roy RODENSTEIN, et Ian SMITH. « Debugging lenses : A new class of transparent tools for user interface debugging ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Making Things Visible, pages 179–187, 1997.

[HUDSON & SMITH 96A] Scott E. HUDSON et Ian SMITH. « SubArctic UI Toolkit user's manual st. Paul release (beta version 0.8e) ». septembre 1996. `http://www.cc.gatech.edu/gvu/ui/sub_arctic/`. GA 30332-0280.

[HUDSON & SMITH 96B] Scott E. HUDSON et Ian SMITH. « Ultra-lightweight constraints ». Dans *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 147–155. ACM Press, 1996.

[HUDSON & STASKO 93] Scott E. HUDSON et John T. STASKO. « Animation support in a user interface toolkit : flexible, robust, and reusable abstractions ». Dans *Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 57–67. ACM Press, 1993.

[HUOT 03] Stéphane HUOT. « Page Web du projet MagLite ». École des Mines de Nantes, équipe CMI, 2003. `http://www.emn.fr/x-info/gina/maglite`.

[HUOT *et al.* 03] Stéphane HUOT, Cédric DUMAS, et Gérard HÉGRON. « Toward creative 3D modeling : an architects' sketches study ». Dans *Proceedings of the 9th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'03)*, septembre 2003.

[HUTCHINS *et al.* 86] E. L. HUTCHINS, J. D. HOLLAN, et D. A. NORMAN. « Direct manipulation interfaces ». Dans NORMAN, D.A. et S. W. DRAPER, éditeurs, *User Centered System Design : New Perspectives on Human-Computer Interaction*, pages 87–124. Lawrence Erlbaum Associates, Hillsdale, NJ and London, 1986.

[INGALLS *et al.* 88] Dan INGALLS, Scott WALLACE, Yu-Ying CHOW, Frank LUDOLPH, et Ken DOYLE. « Fabrik : A visual programming environment ». Dans Norman MEYROWITZ, éditeur, *OOPSLA'88 : Object-Oriented Programming Systems, Languages and Applications : Conference Proceedings*, pages 176–190, 1988.

[ISHII & ULLMER 97] Hiroshi ISHII et Brygg ULLMER. « Tangible bits : Towards seamless interfaces between people, bits and atoms ». Dans *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 de *PAPERS : Beyond the Desktop*, pages 234–241, 1997.

[ISO 85] *The Graphical Kernel System (GKS) : ISO 7942*. International Organization for Standardization, Geneva, Switzerland, 1985.

[ISO 87] « LOTOS - language of temporal ordering specification ». Rapport Technique ISO DP 8807, 1987.

[JACOB 96] Robert J. K. JACOB. « Human-computer interaction : Input devices ». *ACM Computing Surveys*, 28(1) :177–179, mars 1996.

[JACOB *et al.* 99] Robert J. K. JACOB, Leonidas DELIGIANNIDIS, et Stephen MORRISON. « A software model and specification language for non-WIMP user interfaces ». *ACM Transactions on Computer-Human Interaction*, 6(1) :1–46, mars 1999.

[JACOB *et al.* 94] Robert J. K. JACOB, Linda E. SIBERT, Daniel C. MCFARLANE, et M. Preston MULLEN JR.. « Integrality and separability of input devices ». *ACM Transactions on Computer-Human Interaction*, 1(1) :3–26, mars 1994.

[JENSEN 95] Kurt JENSEN. *Coloured Petri Nets : Basic Concepts, Analysis Methods and Practical Use, vol. 2*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1995. ISBN 3-540-58276-2.

[JUSB 03] « jusb : Java USB ». Projet open source, 2003. `http://jusb.sourceforge.net/`.

[KANSY 85] K. KANSY. « 3D extension to GKS ». *Computers and Graphics*, 9(3) :267–273, 1985.

[KNEP *et al.* 95] Brian KNEP, Craig HAYES, Rick SAYRE, et Tom WILLIAMS. « Dinosaur input device ». Dans Irvin R. KATZ, Robert MACK, Linn MARKS, Mary Beth ROSSON, et Jakob NIELSEN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI'95)*, pages 304–309, New York, NY, USA, mai 1995. ACM Press.

[KRASNER & POPE 88] G. E. KRASNER et S. T. POPE. « A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80 ». *Journal of Object Oriented Programming*, 1(3) :26–49, août/septembre 1988.

[KURTENBACH & BUXTON 94] Gordon KURTENBACH et William BUXTON. « User learning and performance with marking menus ». Dans Beth ADELSON, Susan DUMAIS, et Judith OLSON, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems*, pages 258–264, New York, NY, USA, avril 1994. ACM Press.

[KURTENBACH *et al.* 97] Gordon KURTENBACH, George FITZMAURICE, Thomas BAUDEL, et Bill BUXTON. « The design of a GUI paradigm based on tablets, two-hands, and transparency ». Dans *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 de *PAPERS : Handy User Interfaces*, pages 35–42, 1997.

[LACOMME *et al.* 03] Philippe LACOMME, Christian PRINS, et Marc SEVAUX. *Algorithmes de graphes*. Eyrolles, 2e édition, octobre 2003. ISBN 2212113854.

[LANDAY & MYERS 95] James A. LANDAY et Brad A. MYERS. « Interactive sketching for the early stages of user interface design ». Dans Irvin R. KATZ, Robert MACK, Linn MARKS, Mary Beth ROSSON, et Jakob

NIELSEN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI'95)*, pages 43–50, New York, NY, USA, mai 1995. ACM Press.

[LINTON *et al.* 89] Mark A. LINTON, John M. VLISSIDES, et Paul R. CALDER. « Composing user interfaces with InterViews ». *IEEE Computer*, 22(2), Febuary 1989.

[LIPSCOMB & PIQUE 93] James S. LIPSCOMB et Michael E. PIQUE. « Analog input device physical characteristics ». *ACM SIGCHI Bulletin*, 25(3) :40–45, 1993.

[MACKAY 96] Wendy E. MACKAY. « Réalité augmentée : le meilleur des deux mondes ». *La Recherche, numéro spécial 'L'ordinateur au doigt et à l'œil'*, 284, mars 1996.

[MACKAY *et al.* 98] Wendy E. MACKAY, Anne-Laure FAYARD, Laurent FROBERT, et Lionel MEDINI. « Reinventing the familiar : Exploring an augmented reality design space for air traffic control ». Dans *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, volume 1 de *Computer Augmented Environments*, pages 558–565, 1998.

[MACKENZIE & ZHANG 97] I. Scott MACKENZIE et Shawn ZHANG. « The immediate usability of Graffiti ». Dans *Graphics Interface '97*, pages 129–137, mai 1997.

[MALONEY *et al.* 89] John H. MALONEY, Alan BORNING, et Bjorn N. FREEMAN-BENSON. « Constraint technology for user-interface construction in ThingLab II ». Dans Norman MEYROWITZ, éditeur, *OOPSLA'89 Conference Proceedings : Object-Oriented Programming : Systems, Languages, and Applications*, pages 381–388. ACM Press, 1989.

[MANKOFF *et al.* 00] Jennifer MANKOFF, Scott E. HUDSON, et Gregory D. ABOWD. « Providing integrated toolkit-level support for ambiguity in recognition-based interfaces ». Dans Thea TURNER, Gerd SZWILLUS, Mary CZERWINSKI, et Paternò FABIO, éditeurs, *Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI-00)*, pages 368–375, N. Y., avril 1–6 2000. ACM Press.

[MANN 96] Steve MANN. « 'smart clothing' : Wearable multimedia computing and 'personal imaging' to restore the technological balance between people and their environments ». Dans *Proceedings of the Fourth ACM Multimedia Conference (MULTIMEDIA'96)*, pages 163–174, New York, NY, USA, novembre 1996. ACM Press.

[MARSAN *et al.* 95] M. Ajmone MARSAN, G. BALBO, G. CONTE, S. DONATELLI, et G. FRANCESCHINIS. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons, 1995. ISBN 471 93059 8.

[MAXIMILIEN *et al.* 01] Michael MAXIMILIEN, Boyd DIMMOCK, Dan STREETMAN, Brian WEISCHEDEL, Paul KLISSNER, Sunil DUSANKAR, Ron KLEINMAN, Harry MCKINLAY, WINCOR-NIXDORF, Peter DUELLINGS, Roger LINDSJÖ, Steve TURNER, Paul GAY, et Boris DAINSON. « Java API for USB (javax.usb), JSR-80 specification v0.9.0 ». avril 2001. `http://javax-usb.org/`.

[MCCORMACK & ASENTE 88] Joel MCCORMACK et Paul ASENTE. « Using the X toolkit or how to write a widget ». Dans *Proceedings of the USENIX Summer Conference*, pages 1–14, Berkeley, CA, USA, juin 1988. USENIX Association.

[MCDANIEL & MYERS 98] Richard G. MCDANIEL et Brad A. MYERS. « Building applications using only demonstration ». Dans *Proceedings of the 3rd international conference on Intelligent user interfaces*, pages 109–116, 1998.

[MEALY 55] George H. MEALY. « A method for synthesizing sequential circuits ». *Bell System Technical Journal*, 34(5) :1045–1079, 1955.

[MERTZ *et al.* 00] C. MERTZ, J.L. VINOT, et D. ETIENNE. « Entre interaction directe et reconnaissance d'écriture : les gestes écologiques ». Dans *ERGO-IHM 2000*, pages 170–177, Biarritz, France, octobre 2000. CRT ILS & ESTIA.

[MICROÏDS 02] « Syberia ». Microïds, 2002. `http://www.microids.com`, `http://www.syberia.info/`.

[MICROSOFT 03A] « DirectInput C/C++ reference ». Microsoft Corporation, 2003. `http://msdn.microsoft.com/`.

[MICROSOFT 03B] « Transcriber ». Microsoft Corporation, 2003. `http://www.microsoft.com/windowsmobile/resources/downloads/pocketpc/transcriber.mspx`.

[MILGRAM & KISHINO 94] P. MILGRAM et F. KISHINO. « A taxonomy of mixed reality visual displays ». *IEICE Transactions on Information Systems*, E77-D(12), dec 1994.

[MODUGNO 93] Francesmary MODUGNO. « PURSUIT : Programming in the user interface ». Dans *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems – Adjunct Proceedings*, Doctoral Consortium, page 217, 1993.

[MOORE 56] E. F. MOORE. « Gedanken-experiments on sequential machines ». Rapport Technique, 1956.

[MOZILLA 03] « The optimoz project ». The Mozilla Organization, 2003. `http://optimoz.mozdev.org/`.

[MURAKAMI *et al.* 95] Tamostsu MURAKAMI, Kazuhiko HAYASHI, Kazuhiro OIKAWA, et Naomasa NAKAJIMA. « DO-IT : Deformable objects as input tools ». Dans *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 2 de *Interactive Experience*, pages 87–88, 1995.

[MYERS 90] Brad A. MYERS. « A new model for handling input ». *ACM Transactions on Information Systems*, 8(3) :289–320, juillet 1990.

[MYERS 92] Brad A. MYERS. « Demonstrational interfaces : A step beyond direct manipulation ». *Computer*, 25(8) :61–73, août 1992.

[MYERS 93] Brad A. MYERS. « Why are human-computer interfaces difficult to design and Implement ? ». Rapport Technique CMU-CS-93-183, Carnegie Mellon University, School of Computer Science, juillet 1993. `ftp://reports.adm.cs.cmu.edu/usr/anon/1993/CMU-CS-93-183.ps`.

[MYERS 96] Brad A. MYERS. « The Amulet v2.0 reference manual ». Carnegie Mellon University Computer Science Department, février 1996. `http://www.cs.cmu.edu/~amulet`.

[MYERS 98A] Brad A. MYERS. « A brief history of human-computer interaction technology ». *interactions*, 5(2) :44–54, 1998.

[MYERS 98B] Brad A. MYERS. « Scripting graphical applications by demonstration ». Dans *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, volume 1 de *Software Behind the Scenes*, pages 534–541, 1998.

[MYERS & KOSBIE 96] Brad A. MYERS et David S. KOSBIE. « Reusable hierarchical command objects ». Dans Michael J. TAUBER, Victoria BELLOTTI, Robin JEFFRIES, Jock D. MACKINLAY, et Jakob NIELSEN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems : Commun Ground*, pages 260–267, New York, avril 13–18 1996. ACM Press.

[MYERS *et al.* 93] Brad A. MYERS, Richard G. MCDANIEL, et David S. KOSBIE. « Marquise : Creating complete user interfaces by demonstration ». Dans *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, Demonstration Based Systems, pages 293–300, 1993.

[MYERS *et al.* 97] Brad A. MYERS, Richard G. MCDANIEL, Robert C. MILLER, Alan S. FERRENCY, Andrew FAULRING, Bruce D. KYLE, Andrew MICKISH, Alex KLIMOVITSKI, et Patrick DOANE. « The Amulet environment : New models for effective user interface software development ». *IEEE Transactions on Software Engineering*, 23(6) :347–365, juin 1997.

[MYNATT *et al.* 99] Elizabeth D. MYNATT, W. Keith EDWARDS, Anthony LAMARCA, et Takeo IGARASHI. « Flatland : New dimensions in office whiteboards ». Dans Marian G. WILLIAMS, Mark W. ALTOM, Kate EHRLICH, et William NEWMAN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI-99)*, pages 346–353, New York, mai 15–20 1999. ACM Press.

[NG *et al.* 98] Elizabeth NG, Tim BELL, et Andy COCKBURN. « Improvements to a pen-based musical input system ». Dans *OzCHI'98 : The Australian Conference on Computer-Human Interaction*, pages 239–252. IEEE Press, nov 1998.

[NIGAY & COUTAZ 93] Laurence NIGAY et Joelle COUTAZ. « A design space for multimodal systems : Concurrent processing and data fusion ». Dans *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, Voices and Faces, pages 172–178, 1993.

[NIGAY & COUTAZ 95] Laurence NIGAY et Joelle COUTAZ. « A generic platform for addressing the multimodal challenge ». Dans *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1 de *Papers : Multimodal Interfaces*, pages 98–105, 1995.

[NORMAN 81] D. A. NORMAN. « Categorization of action slips ». *Psychological Review*, 1(88) :1–15, 1981.

[NORMAN 02] Donald A. NORMAN. *The Design of Everyday Things*. Basic Books, sep 2002. ISBN 0465067107.

[NORMAN & DRAPER 86] D. A. NORMAN et St.W. DRAPER. *User Centered System design*. Lawrence Earlbaum Assoc., Hillsdale, 1986. ISBN 0-89859-781.

[OPERA 03] « Mouse gestures in opera ». Opera Software ASA, 2003. `http://www.opera.com/features/mouse/`.

[ORR & ABOWD 00] Robert J. ORR et Gregory D. ABOWD. « The smart floor : a mechanism for natural user identification and tracking ». Dans *CHI '00 extended abstracts on Human factors in computer systems*, pages 275–276. ACM Press, 2000.

[PALANQUE & BASTIDE 93] Philippe PALANQUE et Rémi BASTIDE. « Interactive Cooperative Objects : an Object-Oriented Formalism Based on Petri Nets for User Interface Design ». Dans *IEEE / System Man and Cybernetics 93*, pages 274–285. Elsevier Science Publisher, octobre 1993.

[PALANQUE & BASTIDE 97] Philippe PALANQUE et Rémi BASTIDE. « Synergistic modelling of tasks, users and systems using formal specification techniques ». *Interacting with Computers*, 9(2) :129–153, 1997.

[PALAY *et al.* 88] Andrew J. PALAY, Wilfred J. HANSEN, Mark SHERMAN, Maria G. WADLOW, Thomas P. NEUENDORFFER, Zalman STERN, Miles BADER, et Thom PETERS. « The Andrew Toolkit — an overview ». Dans USENIX ASSOCIATION, éditeur, *EUUG Conference Proceedings, Spring, 1988. London, England*, pages 311– ??, Buntingford, Herts, UK, Spring 1988. EUUG.

[PARADIGM 03] « Vega prime ». Multigen-Paradigm, Inc., 2003. `http://www.paradigmsim.com`.

[PARAGON 03] « Penreader ». Paragon Software, 2003. `http://www.penreader.com`.

[PARTRIDGE *et al.* 02] Kurt PARTRIDGE, Saurav CHATTERJEE, Vibha SAZAWAL, Gaetano BORRIELLO, et Roy WANT. « TiltType : accelerometer-supported text entry for very small devices ». Dans *Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST-02)*, pages 201–204, New York, octobre 27–30 2002. ACM Press.

[PATERNÒ & FACONTI 92] F. PATERNÒ et G. FACONTI. « On the use of LOTOS to describe graphical interaction ». Dans *Proceedings of the HCI'92 Conference on People and Computers VII*, Graphics – Design and Techniques, pages 155–173, 1992.

[PATRICK & SACHS 94] Mark PATRICK et George SACHS. « X11 input extension library specification ». X Consortium Standard, X11R6, avril 1994.

[PEARSON & WEISER 86] Glenn PEARSON et Mark WEISER. « Of moles and men : The design of foot controls for workstations ». Dans *Proceedings of ACM CHI'86 Conference on Human Factors in Computing Systems*, Haptic Techniques, pages 333–339, 1986.

[PERLIN 98] Ken PERLIN. « Quikwriting : Continuous stylus-based text entry ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Fast Pen Input, pages 215–216, 1998.

[PETRI 62] C. A. PETRI. « Fundamentals of a theory of asynchronous information flow ». *Proc. IFIP Congress, N-H*, 1962.

[PFAFF 85] G. E. PFAFF, éditeur. *User Interface Management Systems : Proceedings of the Seeheim Workshop*, Berlin, 1985. Springer-Verlag. proceedings of the Workshop on User Interface Management Systems, held in Seeheim, FRG, November 1-3, 1983.

[PIEPER & KOBSA 99] Michael PIEPER et Alfred KOBSA. « Talking to the ceiling : an interface for bed-ridden manually impaired users ». Dans *CHI '99 extended abstracts on Human factors in computer systems*, pages 9–10. ACM Press, 1999.

[PIPER *et al.* 02] Ben PIPER, Carlo RATTI, et Hiroshi ISHII. « Illuminating clay : a 3-D tangible interface for landscape analysis ». Dans Loren TERVEEN, Dennis WIXON, Elizabeth COMSTOCK, et Angela SASSE, éditeurs, *Proceedings of the CHI 2002 Conference on Human Factors in Computing Systems (CHI-02)*, pages 355–362, New York, avril 20–25 2002. ACM Press.

[POLLER & GARTER 84] M.F. POLLER et S. K. GARTER. « The effects of modes on text editing by experienced editor users ». *Human Factors*, 26(4) :449–462, 1984.

[POUPYREV *et al.* 96] Ivan POUPYREV, Mark BILLINGHURST, Suzanne WEGHORST, et Tadao ICHIKAWA. « The go-go interaction technique : Non-linear mapping for direct manipulation in VR ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Papers : Virtual Reality (TechNote), pages 79–80, 1996.

[POYNER 96] Rick POYNER. « Wintab interface specification 1.1 : 16- and 32-bit API reference ». LCS/Telegraphics, mai 1996. `http://www.pointing.com`.

[PREECE *et al.* 94] Jenny PREECE, Yvonne ROGERS, Helen SHARP, David BENYON, Simon HOLLAND, et Tom CAREY. *Human-Computer Interaction*. Addison-Wesley Publishing, Reading, Mass., 1994. ISBN 0-201-62769-8. OCLC 35598754.

[RAISAMO & RÄIHÄ 96] Roope RAISAMO et Kari-Jouko RÄIHÄ. « A new direct manipulation technique for aligning objects in drawing programs ». Dans *Proceedings of the Ninth Annual Symposium on User Interface Software and Technogy*, pages 157–164, New York, novembre 6–8 1996. ACM Press.

[REITMAYR & SCHMALSTIEG 01] Gerhard REITMAYR et Dieter SCHMALSTIEG. « An open software architecture for virtual reality interaction ». Dans Chris SHAW et Wenping WANG, éditeurs, *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-01)*, pages 47–54, New York, novembre 15–17 2001. ACM Press.

[REKIMOTO 96] Jun REKIMOTO. « Tilting operations for small screen interfaces ». Dans *Proceedings of the Ninth Annual Symposium on User Interface Software and Technogy*, pages 167–168, New York, novembre 6–8 1996. ACM Press.

[REKIMOTO 02] Jun REKIMOTO. « SmartSkin : an infrastructure for freehand manipulation on interactive surfaces ». Dans Loren TERVEEN, Dennis WIXON, Elizabeth COMSTOCK, et Angela SASSE, éditeurs, *Proceedings of the CHI 2002 Conference on Human Factors in Computing Systems (CHI-02)*, pages 113–120, New York, avril 20–25 2002. ACM Press.

[ROOSENDAAL & WARTMANN 03] Ton ROOSENDAAL et Carsten WARTMANN. *The Official Blender Gamekit : Interactive 3-D for Artists*. No Starch Press, mars 2003. ISBN 1593270046.

[ROUSSEL 02] Nicolas ROUSSEL. « Videoworkspace : une boîte à outils pour l'exploration de nouvelles techniques de gestion de fenêtres ». Dans *Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine)*, pages 271–274. ACM Press, 2002.

[SALBER *et al.* 99] Daniel SALBER, Anind K. DEY, et Gregory D. ABOWD. « The context toolkit : Aiding the development of context-enabled applications ». Dans Marian G. WILLIAMS, Mark W. ALTOM, Kate EHRLICH, et William NEWMAN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI-99)*, pages 434–441, New York, mai 15–20 1999. ACM Press.

[SALEM & ZHAI 97] Chris SALEM et Shumin ZHAI. « An isometric tongue pointing device ». Dans *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 de *TECHNICAL NOTES : Input & Output in the Future*, pages 538–539, 1997.

[SANTORI 90] Michael SANTORI. « An instrument that isn't really ». *IEEE Spectrum*, 27(8) :36–39, août 1990.

[SCHMUCKER 86] K. SCHMUCKER. « MacApp : an application framework ». *Byte Magazine*, 11(8) :189–193, août 1986.

[SCHWERDTFEGER 00] Richard S. SCHWERDTFEGER. « IBM guidelines for writing accessible applications using 100% pure Java ». IBM Accessibility Center, août 2000. `http://www-3.ibm.com/able/guidelines/java/snsjavag.html`.

[SCHYN *et al.* 03] Amélie SCHYN, David NAVARRE, et Philippe PALANQUE. « Description formelle d'une technique d'interaction multimodale dans une application de réalité virtuelle immersive ». Dans *Actes de la 15ème conférence francophone d'Interaction Homme-Machine (IHM'2003)*, novembre 25–28 2003.

[SENSE8 03] « WorldUp and WorldToolkit : Virtual reality support software ». Sense8 Corp, 2003. `http://www.sense8.com`.

[SHNEIDERMAN 83] Ben SHNEIDERMAN. « Direct manipulation : A step beyond programming languages ». *IEEE Computer*, 16(8) :57–69, août 1983.

[SHNEIDERMAN 98] Ben SHNEIDERMAN. *Designing the User Interface*. Addison Wesley Longman, 3e édition, 1998.

[SMITH 88] David N. SMITH. « Building interfaces interactively ». Dans *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, pages 144–151, 1988.

[SONY 03] « Eye toy ». Sony Computer Entertainment, Inc., 2003. `http://www.eyetoy.com`.

[STARNER & PENTLAND 95] Thad STARNER et Alex PENTLAND. « Visual recognition of american sign language using hidden markov models ». Dans *International Workshop on Automatic Face and Gesture Recognition*, 1995.

[STERIADIS & CONSTANTINOU 03] Constantine E. STERIADIS et Philip CONSTANTINOU. « Designing human-computer interfaces for quadriplegic people ». *ACM Transactions on Computer-Human Interaction*, 10(2) :87–118, 2003.

[STORK & HENNECKE 96] D.G. STORK et M.E. HENNECKE. « Speechreading : An overview of image processing, feature extraction, sensory integration and pattern recognition techniques ». Dans *Proceedings of the 2nd International Conference on Automatic Face and Gesture Recognition (FG 96)*, pages xvi–xxvi. IEEE, oct 1996.

[STRAUSS & CAREY 92] Paul S. STRAUSS et Rikk CAREY. « An object-oriented 3D graphics toolkit ». volume 26, pages 341–349, juillet 1992.

[SUN 98] « Java speech API specification v1.0 ». Sun Microsystems, Inc., 1998. `http://java.sun.com/products/java-media/speech/`.

[SUN 02] « The java accessibility API ». Sun Microsystems, Inc., 2002. `http://www.sun.com/access/`.

[TACTEX 03] « MWK-02 : True multi-touch pressure sensing pad ». Tactex Controls, Inc., 2003. Brevet CA. 2273113. `http://www.tactex.com/`.

[TYSON R. *et al.* 90] Henry TYSON R., Hudson SCOTT E., et Newell GARY L.. « Integrating gesture and snapping into a user interface toolkit ». Dans *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, pages 112–122. ACM Press, 1990.

[UIMS 92] « A metamodel for the runtime architecture of an interactive system ». *ACM SIGCHI Bulletin*, 24(1) :32–37, 1992.

[USB/HID 01] « Universal Serial Bus (USB) device class definition for Human Interface Devices (HID). Firmware specification version 1.11 ». Rapport Technique, juin 2001. `http://www.usb.org/developers/hidpage/`.

[VAN DAM 88] Andries VAN DAM. « PHIGS+ functional description revision ». *Computer Graphics*, 22(3) :125–220, juillet 1988.

[VAN KAMPEN 00] Kurtis J. VAN KAMPEN. « The interface between humans and interactive kiosks ». Input Technologies, LLC, 2000. `http://www.visi.com/~keefner/pdfs/focus1.pdf`.

[VIRTOOLS 01] « Virtools dev ». Virtools SA, 2001. `http://www.virtools.com/`.

[WACOM 03] « Wacom pentools 3.1 ». Wacom Technology Co, dec 2003. `http://www.wacom.com/pentools/`.

[WEIMER & GANAPATHY 89] D. WEIMER et S. K. GANAPATHY. « A synthetic visual environment with hand gesturing and voice input ». Dans *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 235–240. ACM Press, 1989.

[WEISER 91] Mark WEISER. « The computer for the 21st century ». *Scientific American*, pages 95–104, septembre 1991.

[WERTH & MYERS 93] Andrew J. WERTH et Brad A. MYERS. « Tourmaline : Macrostyles by example ». Dans Stacey ASHLUND, Ken MULLET, Austin HENDERSON, Erik HOLLNAGEL, et Ted WHITE, éditeurs, *Proceedings of the Conference on Human Factors in computing systems*, pages 532–532, New York, avril 24–29 1993. ACM Press.

[WORDSPLUS 03] « EZ Keys user manual ». Words+, Inc., 2003. `http://www.words-plus.com`.

[YANG *et al.* 98] Jie YANG, Rainer STIEFELHAGEN, Uwe MEIER, et Alex WAIBEL. « Visual tracking for multimodal human computer interaction ». Dans *Proceedings of the Conference on Human Factors in Computing Systems (CHI-98) : Making the Impossible Possible*, pages 140–147, New York, avril 18–23 1998. ACM Press.

[ZANDEN & MYERS 95] Brad T. Vander ZANDEN et Brad A. MYERS. « Demonstrational and constraint-based techniques for pictorially specifying application objects and boundaries ». *ACM Transactions on Computer-Human Interaction*, 2(4) :308–365, décembre 1995.

[ZELEZNIK *et al.* 91] Robert C. ZELEZNIK, D. Brookshire CONNER, Matthias M. WLOKA, Daniel G. ALIAGA, Nathan T. HUANG, Philip M. HUBBARD, Brian KNEP, Henry KAUFMAN, John F. HUGHES, et Andries van DAM. « An object-oriented framework for the integration of interactive animation techniques ». *Computer Graphics*, 25(4) :105–112, juillet 1991.

[ZHAI 98] Shumin ZHAI. « User performance in relation to 3D input device design ». *Computer Graphics*, 32(4) :50–54, novembre 1998.

UN MODÈLE D'INTERACTION EN ENTRÉE POUR DES SYSTÈMES INTERACTIFS MULTI-DISPOSITIFS HAUTEMENT CONFIGURABLES

Pierre DRAGICEVIC
Doctorat de l'Université de NANTES

Résumé

Aujourd'hui, les applications comme les outils de développement demeurent câblés pour une utilisation exclusive et stéréotypée du clavier et de la souris, et ignorent tout autre paradigme d'interaction. Bien qu'avérés, les problèmes liés à l'adaptabilité en entrée ont encore fait l'objet de très peu d'études. Dans cette thèse, nous proposons un modèle basé sur des *configurations d'entrée*, où des dispositifs d'entrée sont librement connectés aux applications à travers des adaptateurs, et où la traditionnelle file d'événements est remplacée par un paradigme à flot de données réactif. Ce modèle a donné lieu à la boîte à outils en entrées ICON (Input Configurator), qui permet de construire des applications interactives entièrement configurables en entrée, capables d'exploiter des dispositifs et des techniques d'interaction non-standard. Son éditeur visuel permet au développeur de créer rapidement des configurations pour des entrées appauvries ou enrichies, que les utilisateurs avancés peuvent ensuite adapter à leurs besoins.

Mots-clés : Interaction Homme-Machine, Périphériques d'entrée, Techniques d'interaction, Adaptabilité, Accessibilité, Interfaces Post-WIMP.

Abstract

Today's applications and tools exclusively rely on mice and keyboards and use them a stereotyped way, and are still ignoring any other interaction paradigm. Though importance of input adaptability has been widely recognized, its issues have been very little studied. We propose a new input model based on *input configurations*. In this model, input devices are freely connected to applications through adapters, and traditional event mechanisms have been replaced by a reactive data-flow paradigm. Using this model, we developed the ICON (Input Configurator) input toolkit, which allows to build interactive applications that are fully configurable and that can make use of alternative input device and techniques. With its visual editor, developers can rapidly create by direct manipulation configurations for impoverished or enriched input. Power users can then customize those configurations to suit their specific needs.

Keywords : Human-Computer Interaction, Input devices, Interaction techniques, Adaptability, Accessibility, Post-WIMP interfaces.